

A Simulation Tool for Automated Guided Vehicle Systems

by

Bulent Sezen

Project submitted to the Faculty of

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Engineering

in

Industrial and Systems Engineering

Approved:

Dr. M. P. Deisenroth, Chairman

Dr. J. W. Schmidt

Dr. S. C. Sarin

July, 1997

A Simulation Tool for Automated Guided Vehicle Systems

by

Bulent Sezen

Committee Chairman : Dr. M. P. Deisenroth

Industrial and Systems Engineering

(ABSTRACT)

The design and control of an Automated Guided Vehicle (AGV) system is a complex issue and needs planning prior to the implementation of the system. This project provides an analysis tool for evaluation of a variety of AGV systems. A generic AGV simulator has been developed by using the AGV library of a commercial simulation package, SIMPLE++. SIMPLE++ is an object-oriented modeling and simulation environment. Object orientation provides higher development speed, ease of modification, and reduced complexity. The generic AGV simulator is also capable of reading the layout data from an AutoCAD drawing.

Acknowledgements

First of all, I want to thank God for giving me the ability to accomplish this goal. I could not have done this without the help of God. I wish all graduate students who are graduating this year have good positions in the future.

I would like to thank Dr. Mike Deisenroth for the support he has given me as my committee chairman. I appreciate his financial support for my trip to Chicago for SIMPLE++ training. Thanks also to Dr. Schmidt and Dr. Shewchuk for serving on my committee.

I would also like to thank AESOP Corporation for providing us the SIMPLE++ package for this research study. Thanks also to Jerry, the Assistant Manager of the AESOP Corporation, for his helps during the SIMPLE++ training. I would like to thank Turkish Ministry of Education for funding me during this research study.

Finally, thanks to my family for their emotional helps. My parents, my wife, all other family members, and all friends in Blacksburg have always supported me during this research study.

Table of Contents

Chapter I	1
1. INTRODUCTION	1
1.1. Statement of the Objective	1
1.2. AGV System Analysis and Evaluation	2
1.3. AGVs and AGV Systems	3
1.3.1. Basic Vehicle Types	3
1.3.2. Battery Charging	4
1.3.3. Bi-directional Flow	4
1.3.4. Traffic Management	5
1.4. Simulation of AGV Systems	6
1.5. Object-Oriented Simulation	9
1.6. Problem Statement	11
Chapter II	12
2. LITERATURE REVIEW	12
2.1. Automated Guided Vehicle Systems	12
2.1.1. Benefits of an AGV System	13
2.2. Tools for AGV Systems Design Problem	14
2.2.1. Analytical Tools	14
2.2.2. AGV Systems Simulation	15
2.2.2.1. General-purpose Simulation Languages	16
2.2.2.2. Simulation Packages Specific for Manufacturing Systems	18
2.2.2.3. Simulation Software Specifically Created for AGV Simulation	20

2.2.2.3.1. Simulation Studies Based on	
Traditional Programming Approaches	21
2.2.2.3.2. An Object-Oriented AGV Simulation	
Study	24
2.3. Conclusion	25
Chapter III	26
3. REQUIREMENTS FROM A GENERIC AGV SIMULATOR	26
3.1. Introduction	26
3.2. Performance Measures	26
3.3. Production Management and Job Request Types	28
3.4. Independent Workstations	29
3.5. Assembly Operations	30
3.6. Machining Times	30
3.7. Buffers and Interactions	30
3.8. Breakdowns for Machines and AGVs	31
3.9. Dispatching Rules	32
3.10. Battery Charging	33
3.11. Vehicle Routing	34
3.12. Bi-directional flow logic and Multiple Loads	34
3.13. Priorities and Vehicle Assignments	35
3.14. Simulating Wireless AGVs	35
3.15. Traffic Management	36
3.16. Animation	37
3.17. Layout Drawing	37
3.18. Conclusion	38

Chapter IV	39
4. MODEL DEVELOPMENT	39
4.1. Introduction	39
4.2. Object-Oriented Modeling Approach for AGV Systems	39
4.3. SIMPLE++ AGV Application Objects	40
4.4. Existing AGV Application Objects	42
4.4.1. AGV System Manager	42
4.4.2. Production manager	43
4.4.3. Simulation Clock	44
4.4.4. Workstation	44
4.4.5. AGV	45
4.4.6. Part	46
4.4.7. Path Elements	47
4.4.7.1. agvs_straight and agvs_bend	47
4.4.7.2. agvs_br2 and agvs_br3	47
4.4.7.3. agvs_junction2 and agvs_junction3	48
4.4.7.4. AGV Parking station	48
4.4.7.5. Register-line Start	48
4.4.7.6. Terminal Line Branch	49
4.4.7.7. Dock Station	49
4.5. Revisions to the Existing AGV Application Objects	50
4.5.1. Changes Related to AGV System	50
4.5.1.1. Vehicle Types	51
4.5.1.2. Vehicle Assignments	51
4.5.1.3. Vehicle States	52

4.5.1.4. Vehicle Breakdowns	53
4.5.1.5. Shift Work for AGVs	54
4.5.1.6. Pickup/Deposit Interactions	54
4.5.1.7. Spur Option	55
4.5.1.8. Vehicle Speed	55
4.5.1.9. Multiple-load Carrying and Multiple Destinations	56
4.5.1.10. Fixed Vehicle Routing	57
4.5.1.11. Bi-directional Flow Logic	58
4.5.1.12. Traffic Rules	59
4.5.1.13. Dispatching Rules	60
4.5.2. Changes Related to Production System	60
4.5.2.1. Workstation States	60
4.5.2.2. Buffer States	61
4.5.2.3. Job Request Types	61
4.5.2.4. Independent Workstations	62
4.5.2.5. Machining Times	62
4.5.2.6. Job Priority	63
4.5.2.7. Job States	63
4.6. New AGV Application Objects	64
4.6.1. AGV Model Frame	64
4.6.2. Statistics	66
4.6.3. Route Manager	69
4.6.4. Maintenance Station	71
4.6.5. Battery Charge Station	72

4.6.6. Model Builder	73
4.7. Conclusion	75
Chapter V	76
5. ILLUSTRATION OF PROCEDURES FOR USING AGV SIMULATOR	76
5.1. Introduction	76
5.2. An Example AGV Model	76
5.3. Description of the Model	76
5.4. Building the Simulation Model	79
5.4.1. Placing and Connecting the Workstation Objects	81
5.4.2. Setting the Machining Times and Workstation Specifications	82
5.4.3. Setting the AGV System Related Specifications	84
5.4.4. Defining the Part Types, Their Operations Plans and Priorities	85
5.4.5. Setting the Simulation End Time and Initialization	88
5.4.6. Job Arrival Times	89
5.4.7. Setting the Vehicle Specifications	90
5.4.8. Turning the AGV Reset Switch off and Running the Simulation	90
5.5. Conclusion	90
Chapter VI	91
6. STATISTICS AND MODEL VALIDATION	91
6.1. Introduction	91
6.2. Validation of the Outputs of the Example Model	91
6.2.1. Job Statistics	91

6.2.2. Vehicle Statistics	94
6.2.3. Machine Statistics	97
6.2.4. Zone Statistics	98
6.3. Validation of the Capabilities of the Simulator	102
6.3.1. Multiple-Load Carrying Capability	103
6.3.2. Battery Charging	104
6.3.3. Vehicle Breakdowns	104
6.3.4. Job Arrival Types	107
6.3.5. Vehicle Assignments	109
6.3.6. Traffic Rules	112
6.4. A Larger Model	113
6.5. Conclusion	116
Chapter VII	117
7. CONCLUSIONS AND RECOMMENDATIONS	117
7.1. Conclusions	117
7.2. Recommendations	118
Bibliography	120
Appendix I : TRANSFERRING CAD DRAWINGS INTO SIMPLE++	128
Appendix II : INPUT DATA FOR THE LARGE EXAMPLE MODEL	136
Appendix III : NEW METHODS IN THE MODIFIED AGV LIBRARY	138

List of Figures

Figure 4.1. The SIMPLE++ AGV objects library	
41	
Figure 4.2. The modified AGV objects library	65
Figure 4.3. The structure of the Stats object	66
Figure 4.4. The structure of the RouteMgr object	69
Figure 4.5. The structure of the Maintenance object	71
Figure 4.6. The structure of the bat_charge_station object	72
Figure 4.7. The structure of the Build object	74
Figure 5.1. The layout for the example AGV model	77
Figure 5.2. The appearance of the sample AGV model after translation	80
Figure 5.3. The appearance of the sample model after placing stations	82
Figure 5.4. The dialog window of a station object	83
Figure 5.5. The dialog window of the agvs_manager object	84
Figure 5.6. The dialog window of the AGV object	85
Figure 5.7. The dialog window of the agvs_station object	86
Figure 5.8. The dialog window of the ProdMgr object	86
Figure 5.9. The table for setting the part type information	87
Figure 5.10. The dialog window of the EventController object	
88	
Figure 5.11. The dialog window for setting the part arrival times	89
Figure 6.1. The dialog window of the Stats object	92
Figure 6.2. The histogram for the total time spent in the system	93
Figure 6.3. The histogram for the total vehicle response time at station	95

Figure 6.4. The final vehicle utilization report	96
Figure 6.5. The final machine utilization report	97
Figure 6.6. A sample zone statistics report	98
Figure 6.7. A sample of report showing the number of blockages	100
Figure 6.8. The example model with a new battery charging station	105
Figure 6.9. The new configuration of the model with the maintenance	106
Figure 6.10. Definition of custom attributes for agv-1	107
Figure 6.11. Illustration of activating type B breakdowns	108
Figure 6.12. The table containing part ordering information	109
Figure 6.13. Assigning vehicles to serve a specific dock station	110
Figure 6.14. Assigning vehicles to serve specific part types	111
Figure 6.15. The dialog window of a junction element	112
Figure 6.16. Schematic diagram of the AGV layout for sample model	113
Figure 6.17. The corresponding SIMPLE++ model for the large model	114
Figure A.1. The initial CAD drawing of the example model	130
Figure A.2. The drawing after the insertion of intersection entities	131
Figure A.3. The complete list of intersection types and their names	132
Figure A.4. The appearance of drawing after inserting the control points	133
Figure A.5. The drawing after insertion of p/d points	134
Figure A.6. The drawing after the insertion of direction entities	135
Figure A.7. Setting the interface file in dialog window of the Build object	135

List of Tables

Table 3.1. Comparison of existing simulation programs for AGV systems	38
Table 5.1. Length between location points in the layout	79
Table 6.1. Idle vehicle status report	97
Table 6.2. Total number of passes from each segment	99
Table 6.3. Total number vehicle entries, loads, and unloads at stations	101
Table 6.4. Total number of vehicle passes at intersections	101
Table B.1. Time between part arrivals for part types in the large model	136
Table B.2. Operation sequences for the part types in the large model	137

Chapter I

INTRODUCTION

1.1. Statement of the Objective

The objective of this study was to develop an analysis tool for evaluation of a variety of automated guided vehicle (AGV) systems. A generic simulation package has been created by using SIMPLE++, an object-oriented software package for integrated modeling, simulation, and animation. The primary target of the simulation package will be prospective academic researchers who study design and control issues of AGV systems. Additionally, AGV suppliers and companies that are using AGVs may also use the package as an unbiased simulator.

The package is flexible in terms of the definition of the system, aspects of the design and control of the system, and general extensions. First, flexibility in the definition of the system means the package can simulate a wide range of AGV systems without any configuration limits. For example, bi-directional flow, multiple-load carriers, wireless vehicles, different battery charging techniques, and many other features of an AGV system can be simulated by using the package. Second, the flexibility in the design and control of AGV systems is the capability of adding new rules and methodologies into the original code. For example, a new vehicle dispatching rule may be needed in the future. It will be easy to modify the code in order to add the new rule. Third, general extensions such as adding new objects, defining new classes, or modifying the structure of the original code, will be possible if the user has the knowledge of object oriented programming and the built-in language of SIMPLE++.

1.2. AGV System Analysis and Evaluation

Analysis of AGV systems can be separated into design and operational problems. The design phase covers the problems related to the design issues of AGV systems such as pickup and deposit point locations, AGV fleet size, speed of vehicles, AGV maintenance plan, types of vehicles, AGV load capacity, load transfer mechanism, etc. The operational phase starts with the actual use of AGV system. In this phase, the control strategies of an AGV system are examined. For example, rules for dispatching the vehicles, vehicle priority assignments, traffic control logic, job assignments to different vehicle types, etc. are evaluated in the control phase.

Typical system objectives in the design and control of AGV systems are:

1. Evaluation of the feasibility of an AGV system -- The performance of AGV systems may be compared with the performance of other types of material handling systems, e.g. conveyor systems.
2. Evaluation of the dispatching rules -- The performance of a system may be tested under a variety of vehicle dispatching rules.
3. Elimination of traffic problems -- Traffic control strategies at intersections must be properly designed to avoid vehicle collisions in the system.
4. Maximizing the throughput -- The number of finished jobs per unit time may be increased by modifying the AGV system configurations.
5. Maximizing the vehicle utilization -- The minimization of the times of vehicle waiting, breakdowns, and empty travels may be the objective of a study. The utilization of vehicles directly affects the determination of the AGV fleet size.

6. Minimizing the inventory level -- Inventories must be kept at the right places and at the right time. The selected vehicle dispatching rule may change the inventory level drastically.
7. Minimizing the transportation costs -- The minimization of the number of vehicles in the system and the maintenance costs must be considered at the design phase of an AGV system.
8. Maximizing the space utilization -- The system layout must be arranged by considering the current space constraints and possible extensions to the manufacturing layout.

1.3. AGVs and AGV Systems

An AGV system is an advanced material handling system that involves a driverless vehicle which follows a path (virtual or guided) and is controlled by a computer or microprocessor [Hammond, 1986]. A typical AGV consists of the frame, batteries, electrical system, drive unit, steering, precision stop unit, on-board controller, communication unit, safety system, and work platform.

AGV systems are mainly used for distribution of materials in warehouse environments, and movement of material to and from production areas and storage areas in manufacturing facilities. Some other applications of AGV systems in non-manufacturing environments include, but are not limited to, delivering mail, messages, and packages in offices, and delivering meals and laundry in hospitals.

1.3.1. Basic Vehicle Types

Types of AGVs can be categorized as towing vehicles, pallet trucks, and unit-load carriers. *Towing vehicles* pull a series of trailers that are attached to the vehicle. The trailers are attached to and detached from the vehicle manually at the stations. The vehicle does not have lifting capabilities nor a transfer

mechanism. It can be used for any type of load. *Pallet trucks* are used for palletized loads and can have high lifting capabilities. They can pick up and deposit loads at the floor level. *Unit-load carrier* may carry single or multiple loads on their deck. Some are capable of traveling sideways. The transfer mechanism of the carrier can be either an active or passive conveyor, such as a roller, belt, or chain conveyor, or it may be a lift/lower deck.

1.3.2. Battery Charging

AGVs are battery-powered vehicles. Battery charging is usually accomplished by either an opportunity charging technique or a full-cycle charging technique. The opportunity charging technique charges the batteries while the vehicle is stopped at a station. The full-cycle charging technique requires that the AGV be taken out of service by directing it to a specific charging area. In this technique, the battery can be charged on the vehicle either manually or automatically [Hammond, 1986]. Additionally, it is possible to exchange the depleted battery for a spare battery which has already been charged.

1.3.3. Bi-directional Flow

If a vehicle can travel either in a forward or reverse direction along the path under automatic control mode (not manual), it is called a “bi-directional vehicle”. However, when the path is said to be a “bi-directional path”, this means the vehicles are allowed to move in both directions on the path. This kind of flow is called “bi-directional flow”. Many AGVs are bi-directional when operated manually, but they are not allowed to move in both directions in the path.

Because bi-directional flow in a system increases the complexity of traffic control, most industrial applications avoid using bi-directional path. In general, it is only applied to a small section of the AGV path.

For example, if there is a terminal line on the path, the vehicle must be able to travel in both directions on this terminal line. Only this part of the path is bi-directional and the rest is uni-directional.

1.3.4. Traffic Management

Traffic management is the ability of the AGV system to avoid collisions and to maximize vehicle flow efficiency. Two types of traffic management methods are forward sensing and zone control. In the *forward sensing method*, there is no central control and vehicles have the capability of sensing the vehicles traveling in front of them. An on-board system built into the vehicle is capable of sensing objects that are located within a certain distance ahead of the vehicle, e.g. 10 feet. Any obstacle within this distance causes the vehicle to stop. For this reason, the forward sensing method may be used only in non-complicated, simple loop AGV systems.

In the *zone control method*, the layout is divided into zones. Only one vehicle is permitted in a zone at any time. A zone may include multiple pickup and/or deposit points in it. It is also possible to have a zone without any pickup or deposit points. Zone control can be accomplished in three different ways [Koff, 1987].

1. *Distributed zone control* requires individual control of each zone. A zone control box is located in each zone and will stop a guided vehicle from entering the zone if another vehicle already occupies that zone.
2. *Central zone control* applies the same concept of zone controlling, only through the use of zone communication points that are controlled by a central controller. When a vehicle comes to a zone entrance, it communicates with the central controller through the zone communication point and waits for the permission to enter the zone.

3. *On-board control* means that vehicles communicate with each other without a central zone controller. When a vehicle enters into a new zone, it changes the information it is transmitting to indicate its new position. The advantage of the on-board control is that failure of the controller is limited to a single vehicle and not to a central controller failure which affects all the vehicles in the system.

1.4. Simulation of AGV Systems

Simulation is a technique that is used to test and measure the behavior of a specific system under given set of conditions. To evaluate alternative design and control strategies of AGV systems, it is required to estimate the system behavior under these strategies. Simulation enables the user to create a computer model of the actual AGV system that can be used to measure the effects of design and control related changes to the system.

One advantage of simulation is that a system can be tested without existing in the real world. This may be the case if a company attempts to evaluate the feasibility of an AGV system before making an investment. Simulation can also be used to test the effects of a layout change in an existing AGV system. Another advantage of simulation is its speed of execution. Long lasting operations can be simulated in a few seconds. This may be important if a dynamic AGV control system or a dynamic shop floor control system is utilized.

Simulation is the most suitable technique when random behavior exists in a system. Vehicle breakdowns, machine breakdowns, job requests, and many other AGV system-related events are random in nature. This randomness in AGV systems brings complexity in the solution of the design and control problems. Simulation is the most powerful and widely-used tool in the analysis of such complex systems. For many AGV systems, simulation is not an option, but a requirement [Hammond, 1986].

One of the most important part of the simulation modeling process is the definition of inputs. The inputs for the simulation model of an AGV system can be partitioned into two types: AGV system data and manufacturing system data. Both AGV system data and manufacturing system data will be discussed in the following paragraphs.

AGV system data consists of layout-related inputs, vehicle-related inputs, and control-related inputs. The two *layout-related inputs* are the guide path layout and locations of specific points on the path. These specific points are control points, pickup points, deposit points, parking area, and charging area. A *control point* is a point prior to entering an intersection or prior to a zone that takes only one vehicle at a time. A *pickup point* is a point where an AGV gets a load from a workstation. A *deposit point* is a point where an AGV delivers a load to a workstation. A *parking area* is a place where AGVs wait when they are idle or broken. A *charging area* is a place where AGVs charge/change their batteries. The travel distances between each point are also included in the layout-related data.

The *vehicle-related inputs* are AGV speed, number of AGVs in the system, vehicle loading / unloading times, load capacities of the vehicles, and battery charging / changing data. Speed of a vehicle may differ according to the current status of the vehicle. In general, a loaded vehicle moves slower than an empty vehicle. Additionally, an AGV traveling on a straight path is faster than an AGV traveling on a curved path. An AGV system may employ different types of AGVs for different types of tasks. The number of vehicles must be specified for each type of vehicle in the system. The load capacity of a vehicle is given as the number of units it can transport at the same time. A unit can be a part, a container (e.g. a bin), or any other type of entity that needs to be transported by the vehicle. Battery charging / changing data consists of the maximum battery charge capacity, battery consumption rate (amperes per hour), battery charging current (amperes per hour), battery charge safety level, current battery charge

level, and battery exchange time. Battery charge safety level is a specific charge level where the vehicle is sent to the charging area for a battery charging or changing operation. Battery exchange time is the time for replacing the old batteries with new ones (if the battery exchange technique is utilized).

The *control-related inputs* are collision avoidance protocols, dispatching rules, idle vehicle strategies, vehicle assignments to specific tasks, and the type of battery charging / changing technique. Collision avoidance protocols ensure that no two vehicles occupy the same position at the same time. One method is to use the zone controlling technique. Another method may be setting fixed distances between vehicles. In this method, a vehicle stops if there is another vehicle within the specified distance ahead of it. Dispatching rules are categorized as vehicle-initiated and equipment-initiated dispatching rules. These are the rules for dispatching a vehicle to perform a task and will be explained in detail in the following chapters. Idle vehicle strategies are for determining the question of “what an idle vehicle does?” An idle vehicle may be sent to the parking area, to a specific workstation, or it may be turning around a loop on the AGV path. Vehicle assignments to specific tasks are done to simplify the control of AGV systems. If there are such assignments, the assignment specifications, such as the assigned vehicles and their destination workstations, must be defined. The type of battery charging / changing technique can be normal battery charging, battery exchange operation, opportunity charging, or a mix of these techniques.

Manufacturing system data consists of processing times, workstation operations list, machine breakdown data, buffer sizes, route for each part type, job priorities, ordering data, and job (or order) arrival times data. Processing times are either part specific or machine specific. The actual processing time, setup time, and waiting time in the buffer are included in the processing times data. The workstation operations list contains the names of operations to which the workstation is assigned. Machine breakdown data include the time between

breakdowns and the repair time for each machine. Buffers are placed before and after workstations. The size of a buffer can be set to a limited number, or it can be unlimited. Buffer size data could be under the category of AGV system data, however, it has been placed under the manufacturing system data because manufacturing systems may have buffers without having an AGV system. The route for each part type, or job, must be given as an operation-sequence table. The simulator must also know which workstations are assigned to which operations. By this knowledge, a job is routed to proper workstations according to its operation-sequence list. Job priorities are required if some job types are more important than others. For example, a rush order in the system will have the priority of having the vehicle first for a transport. The ordering data is a list of orders for part types. Ordering time and the number of units requested for each order are entered by the user. Job arrival times data describe the time between job arrivals to a workstation. It can be defined as a randomly generated statistical distribution value.

These inputs to the simulation model must be well matched with the specifications of the real AGV system. For example, if the simulated AGV system utilizes a first-in-first-out rule as the vehicle dispatching rule, the simulation model must be set for the exact same rule. Otherwise, the model would not give the same results with the real system.

1.5. Object-Oriented Simulation

Today's manufacturing environments are constantly changing due to rapid development of the technology and demand. Simulation tools for such environments need to be flexible, easy to use, and reliable. Object-oriented simulation is a technique that offers modularity, portability, extensibility, and reusability [King and Kim, 1995].

In object-oriented programming approach, objects send messages to each other to perform tasks. A message invokes a certain “method”, or action, in the class to which the object belongs. A single message can produce different reactions when applied to different class objects. This is called “polymorphism.” Objects are hierarchically organized in subclass relationships. The characteristics of a superclass object are inherited by its subclass objects. This feature is called “inheritance.” Inheritance encourages the design of classes by creating hierarchical set of classes, or “building blocks.” A new class can be easily constructed from an old class which already has some of the desired methods and characteristics. This feature supports the reusability of the simulation code. If there are differences between a superclass object and its subclass object, those differences must be defined in the subclass object’s characteristics.

An AGV and a part are both movable units. It is wise to define a class called “Movable Unit” and then define AGV and Part as the subclasses of the class Movable Unit. An AGV and a Part will both share some attributes of their superclass. As an example, they will both know their location in the system at any time. On the other hand, some attributes will not be inherited from the superclass Movable Unit.

For example, if we want to assign a Move function to both classes, that function will do different things for an AGV and for a Part. The part will only move itself from a location to another, where the AGV will move both itself and the Part(s) on it at the same time. Therefore, the Move function must be defined separately for each subclass of the superclass Movable Unit.

Object orientation provides higher development speed, ease of modification, and reduced complexity. It would be easy to add a new movable unit to the described model. For example, if a man powered carrier is going to be added to the simulation model, it can be easily declared as a new subclass of the class Movable Unit.

1.6. Problem Statement

Associated with the developing research thrust in AGVs at Virginia Polytechnic Institute and State University, a new development in AGV system design and control concepts is needed to evaluate performance of new ideas in this area. Currently, there is not an AGV system available in Virginia Polytechnic Institute and State University. However, the simulation method can be used to test the new design and control strategies that are applicable to AGV systems. The future researchers who will develop such new ideas may need a test-bed simulation tool for analyzing their approaches.

Rather than having each researcher develop their individual AGV models for specific AGV systems, it is proposed that a simulation tool be developed that can be used by future researchers to test their design and control approaches. The development of that generic simulator is the subject of this research study. The task of building new models is left to those future researchers.

Chapter II

LITERATURE REVIEW

2.1. Automated Guided Vehicle Systems

The first AGV application was for transporting groceries in a warehouse. [Hammond, 1986]. According to statistics in 1989 [Gould, 1990], AGV system installations with respect to their application types were profiled as following.

1. JIT delivery systems	56%
2. FMS/FAS transfer system	13%
3. Storage load transfer, non-AS/RS	12%
4. AS/RS interface	8%
5. Progressive assembly	7%
6. Mini-load AS/RS interface	1%
7. Other	3%

AGVs can be used in two different ways [Hammond, 1986]. The first approach is to attach a workpiece to the AGV having all manufacturing processes done while the AGV carries the workpiece from station to station. In this approach, the AGV is freed only after all the processes for the workpiece are completed. In other words, the vehicle is assigned to the workpiece until its all operations are finished. The second approach is to use vehicles only for moving the workpieces from one station to another. In this approach, the vehicle is assigned to the workpiece only for a single trip. In the first approach, the number of vehicles required is significantly greater than in a normal AGV system. General Motors Company is the pioneer of such an assembly system built in the U.S. with 185 unit-load carriers.

AGV guidance techniques are wire guidepath technique, optical guidepath technique, and off-wire guidance technique. In the wire guidepath technique, wires with varying frequencies are buried in the floor. AGVs select a path at a control point according to the assigned frequency. In the optical guidepath technique, an AGV focuses a beam of light on a reflective tape or a painted strip and follows the path by measuring the amplitude of the reflected light.

The disadvantages of wire guidepaths and optical guidepaths have caused the development of off-wire guidance techniques such as laser triangulation, floor-grid referencing, and gyroscopic guidance. The advantages of these techniques are that there is no need for floor cutting or painting, and the guidepaths are easily modified.

2.1.1. Benefits of an AGV System

According to case studies of AGV applications provided by the Material Handling Institute (1993), benefits of building and using AGV systems include:

1. Labor costs saving,
2. Better schedule of WIP,
3. Flexible material handling,
4. Effective inventory control,
5. Greater quality assurance and safety,
6. Increased production,
7. Improved utilization of space, and
8. Flexible routing.

2.2. Tools for AGV Systems Design Problem

Tools used in AGV system design can be classified in two main categories: analytical tools and simulation-based tools. Analytical tools are mathematical techniques such as queuing theory, integer programming, heuristic algorithm, and Markov Chains. A number of analytical approaches to the design of AGV systems have been proposed in the literature.

2.2.1. Analytical tools

Tanchoco et al. [1987] compared the effectiveness of CAN-Q, an analytical model based on queuing theory and used for analyzing work flows through a manufacturing system, with a simulation-based model built in AGVSim [Egbelu and Tanchoco, 1982]. CAN-Q underestimated the actual number of vehicles required. Their analysis indicates that the results obtained through CAN-Q provide a good starting point for the simulation study.

Bozer and Srinivasan [1991] introduced the concept of 'tandem configuration' to the design of AGV systems. The tandem configuration is based on partitioning all of the workcenters into non-overlapping, single vehicle closed loops. It offers less complicated control systems, but has less tolerance for vehicle breakdowns and requires additional floor space. The authors also developed an analytical model to estimate the throughput capacity of a single vehicle in a closed loop.

Mahadevan and Narendran [1993] developed an analytical model for estimation of the number of AGVs. They suggested to start with rough-cut analytical methods, followed by the use of sophisticated mathematical models and then to apply simulation if the level of complexity of the AGV system was high. As the number of parts in the system increases, the problem becomes intractable and needs to be analyzed by simulation method.

Johnson and Brandeau [1993] modeled an AGV system as a queuing system and the design model was formulated as a binary integer program with non-linear waiting time constraints. They then developed two different enumeration algorithms to solve the analytical model.

Srinivasan et al. [1994] proposed a general-purpose analytical model to compute the throughput capacity of a trip-based material handling system, such as an AGV system. They stated that the larger the number of devices in the system, the more error they would expect from their analytical approximation method.

Analytical models are generally based on steady-state flow systems [Tanchoco, 1994]. Therefore, analytical techniques may fail when they are applied to real industrial cases. These techniques may give inaccurate estimates under random environments. In conclusion, analytical techniques may be best suited for obtaining initial estimates in the design of an AGV systems [Egbelu, 1987].

2.2.2. AGV Systems Simulation

Simulation software that can be used for AGV system simulation can be grouped in three categories [Tanchoco, 1994]:

1. General-purpose simulation languages (e.g. SLAM II, SIMAN IV, etc.),
2. Simulation packages designed for the general simulation of manufacturing systems (e.g. SIMPLE++, AutoMod II, ProModel, SIMFACTORY II.5, etc.), and
3. Simulation software specially created for analyzing AGV systems by using general programming languages such as C, FORTRAN, BASIC, LISP, etc.

2.2.2.1. General-purpose Simulation Languages

Several AGV system simulation models have been developed using general-purpose simulation languages such as SLAM II [Pritsker, 1995], SIMAN [Pegden et al., 1990], SIMNET II [Taha, 1992], and GPSS/H [Henriksen, 1989].

Seifert et al. [1995] developed a discrete-event simulation model written in SLAM II to analyze the operation of an AGV system under a variety of vehicle routing strategies. Their model handled multiple layouts and pedestrians in the system. It was a mixed-language model that was written in SLAM II with event-processing functions written in the C programming language. A specific performance measure was employed by their simulation model. It was the difference between the AGV's actual travel time and the corresponding theoretical travel time of the AGV with respect to its speed and the travel length. They also employed a concept of "hierarchical simulation." The performance of each alternative route in the subsimulations was used to make routing decisions in the main simulation.

Ulgen and Kedia [1990] built a simulation model using SLAMSYSTEM to analyze the main effects of the design and operational variables on the performance of a cellular assembly system employing AGVs. The factors and alternatives considered in their model were alternate track layout designs, the effect of scheduling rules, and the effect of different cycle time ratios. The measure of the system performance was the average throughput per shift. They stated that the simulation was easy to implement ,especially, the scheduling rules in the SLAMSYSTEM.

Lee et al. [1990] developed a simulation model in SIMAN to evaluate the design of AGV systems for a particular layout. They assumed that the optimal manufacturing layout was created by using analytical models. Their system

performance measures were defined as the utilization of AGVs, average number of jobs waiting for an AGV, throughput time, and the average waiting time at intersections.

Takakuwa [1993] created a simulation model in SIMAN to measure the cost effectiveness of large scale AS/RS-AGV systems based on the number of AGVs to install. First, the overall layout of the system was determined, and then, specifications of the system such as number of AGVs, number of conveyors, the buffer size on each conveyor, and so on, were defined. Their main system measure was the total flow time.

Lee [1996] developed a discrete-event simulation model in SIMAN to evaluate the performance of a number of composite AGV dispatching rules which could be implemented in manufacturing or assembly systems. The number of AGVs needed were determined based on a preliminary simulation study. The performance measures were the throughput, flow time, and the WIP level.

Choi et al. [1994] built a discrete-event simulation model with SIMAN to evaluate a traditional “multiple vehicle” layout and a tandem layout with single vehicle loops. The design parameters included the number of vehicles, AGV speed, job assignment rule, and job arrival distribution. The measures of the system performance were the number of jobs completed, the average vehicle utilization, the average job waiting time at the staging area, and the total blocking time of vehicles.

McHaney [1995a] developed a simulation model by using GPSS/H simulation language showing that the battery constraint in an AGV system could have significant impact on its operation and simulation output. Simulating battery

charge in an AGV system required additional input data on battery limitations and battery charging requirements.

Although material handling features have been added to the general-purpose simulation languages, such as SLAM and SIMAN, these features do not provide sufficient flexibility to simulate the great diversity of many different material handling systems [King and Kim, 1995]. Simulation of AGV systems can be more easily accomplished by starting with simulation packages specifically developed for manufacturing systems.

2.2.2.2. Simulation Packages Specific for Manufacturing Systems

The second category of simulation software includes some general-purpose manufacturing simulation systems such as AutoMod II, ProModel, XCELL+, SIMFACTORY II.5, SIMPLE++, etc.

Prasad and Rangaswami [1988] developed a graphic simulation model of an integrated semiconductor sort, assembly and test facility by using AutoMod as the primary tool. AutoMod is a 'macro' language which is based on the GPSS simulation language. Two different AGV control systems, a global control system versus a local control system, have been analyzed with the simulation model. They stated that AGV bottlenecks, congestion and deadlocks could be easily identified by using AutoMod's animation feature.

Quinn [1985] created a simulation-based system with AutoMod for development and testing of AGV control software. Data from a CAD system was used to describe the guide path. Output from the model was interfaced to an emulator that imitated the network protocol to the controller for testing of the controller software. A generic blocking scheme was used in the model. Quinn [1985] stressed the fact that AGV vendors had unique blocking designs of their own. The blocking design in the simulation model had to be close to the implementation design.

Jayaraman [1993] developed an AGV system design for a company manufacturing antilock breaking systems by using ProModel. He compared manual transportation with AGV transportation. Input data to the system included a file containing the plant layout in AUTOCAD drawings of each assembly cell, forecast requirements, processing times, material handling information, and the bill of materials. The system throughput and the AGV utilization were used as the measure of system performance.

Dewsnup and Bollenbach [1995] discussed the use of ProModel for Windows for modeling AGV systems. They studied two separate but overlapping AGV systems. Their model needed a special intersection logic to avoid collisions. The objective was to determine the number of AGVs and to identify control logic for the AGV system.

Some other commercially available simulation software packages specifically developed for manufacturing systems include MAST [CMS Research, 1990], MODELMASTER [Miller, 1987], MUSIK [Warnecke et al., 1986], and ARENA 2.0, MicroSaint, Taylor II, and Extend+Manufacturing [Feare, 1996]. Three of these software packages are discussed in the following paragraphs.

Extend is a graphical, interactive, general-purpose simulation software package. It provides libraries of iconic blocks which can be used to build models. Its built-in language, modl, can be used to create new blocks. For manufacturing applications, Extend+Manufacturing is offered as a separate package to the customer. Extend is a good product for simulation, but it is not good if programming is needed. The modl language has been reported to be difficult to understand due to lack of information about the language [Marsh, 1995]. In addition, Extend provides limited support for animation.

Taylor II is a simulation software package for manufacturing and business decision making. It provides 2D and 3D animation of the model. The modeling elements for manufacturing are buffer, machine, transport, AGV, conveyor,

warehouse, etc. The language interface of Taylor II, TLI, can be used to create more complex simulation models [Taylor Times, 1996]. Taylor II has been specifically designed for easy simulation of manufacturing systems without having the knowledge of programming. Although TLI is provided with the package to the user, programming with TLI is not encouraged.

SIMPLE++ is the simulation software package that will be used to create the generic AGV simulation tool in this research. It has graphical user interface with basic modeling blocks, which are called “physical objects”. Some of these physical objects are processor, conveyor, vehicle, track, line, entity, container, etc. There are also AGV application objects that are provided as an application specific model. SimTalk is the built-in language for developing application specific code in SIMPLE++. It is based on the object oriented language Eiffel.

2.2.2.3. Simulation Software Specifically Created for AGV Simulation

There are some simulation software packages developed specifically for analyzing AGV systems. In this section, we will briefly review these AGV simulation programs. The initial part of the section will describe codes developed by traditional programming methodologies while the later part of the section identifies efforts that are object-oriented approaches.

2.2.2.3.1. Simulation Studies Based on Traditional Programming Approaches

Egbelu and Tanchoco [1982] developed AGVSim, a simulation package for designing AGV systems. The package was developed in the FORTRAN language. It was a tool for analyzing, planning, and designing AGV systems. In this package, a network was modeled as a collection of nodes and arcs. Only the unidirectional flow pattern was considered. Sinriech and Tanchoco [1992] used AGVSim to evaluate the performance of a single loop guide path under

different dispatching rules. They stressed the impact of empty vehicle flow on the system performance.

AGVSim consists of two separate routines. The first routine calculates the shortest path between pairs of points in the network. The second routine is the main simulator which executes the simulation and reports the results. The input data can be read from an ASCII file into the simulation model. AGVSim provides support programs to enter and remove data. It is possible to add new dispatching rules by using the template routines.

Andersson [1985] created SattControl, an AGV system simulation package, which was used as a tool for planning and testing of an AGV route layout. Entered parameters to the package included AGV track layout, number of pallets per hour, number of AGVs, loading/unloading times, AGV speed (separate for loaded and unloaded vehicles), alternative routes, etc. The layout was drawn in a CAD program. Some of the output statistics were waiting time at a certain point, the number of jobs performed by each AGV, and the average time the AGVs have been idle.

Araki et al. [1987] developed a simulator that could handle 68 kinds of AGV path patterns including station, entrance, exit, direct line, curve, T-cross, etc. The simulator consisted of path editor, numerical data input editor, shortest path calculator, simulation executor, animation section, and results output section. The shortest path calculator and the simulation executor were programmed in FORTRAN, and the rest were programmed in BASIC. The simulation executor had the initial setting, time management, AGV dispatch management, machine tool management, and data management functions.

Schulze and Rosenbach [1987] built MATSIM, a special simulation software-tool with a module library, for material flow systems. MATSIM input

data included route, vehicle data, priority strategies, and sequence of processing. Output data were the total output of the system, number of loaded and empty travels, blocking time, battery charge time, vehicle waiting time, and temporal using of workstations.

Ozden [1988] developed a discrete-event simulation program by using LISP to investigate the effect of key factors, such as number of pallets, number of vehicles, and carrying capacity, of multiple-load-carrying AGVs on the overall performance of a FMS. The simulation program provided an animated color view of the FMS operation upon the user's request. He also stated that the predefined functions of LISP enabled the user to design simulation models with modularity similar to special-purpose simulation languages.

Mosca et al. [1991] constructed and utilized a transporting network simulator inside a large dimensions fruit and vegetable market utilizing AGVs. The discrete and stochastic simulator was programmed in FORTRAN 77 language. Its general objectives were to plan choices to minimize investment costs, to minimize service times, and to maximize AGV utilization in the system.

Gaskins and Tanchoco [1989] presented AGVSim2, a discrete-event vehicle system simulator, for evaluating the control strategies of real-time free-ranging vehicle controllers. This simulator was directly linked to the supervisory controller software so that it could be used to test the intelligent AGV supervisory controller. It was programmed in the C programming language.

AGVSim2 allows for bi-directional flow, multiple vehicle types, and multiple loads on a vehicle. A graphic animation module is also a part of the software package. For free-ranging vehicles, virtual control points are used. The only

difference between regular and virtual control points is that regular control points have attributes that are stored in the simulator's data structures. Otherwise, they are both treated the same way.

Dutt [1991] developed GVSIM, a generic C-based discrete-event simulation package, for modeling alternative AGV system configurations. Various control concepts in the design of guided vehicle systems could be tested by the generic program. It also extended the shortest path concept to develop the quickest path concept for reduction of bottlenecks. In addition, vehicle allocation was not limited to only idle vehicles. Input by the user were the layout, number of vehicles, and generation of vehicle requests. Example AGV systems were simulated using both SLAM and GVSIM and the results were compared for model validation.

Wilson [1992] developed a C-based discrete-event simulation framework for analyzing AGV systems. The functions in the framework were based on the discrete simulation portion of CBST [Khan, 1991], a C-based Simulation Toolkit. The functions of CBST form a modular toolkit (C functions) including executive control, variable initialization, list management, entity management, resource management, random variate generation, data collection, and reporting.

Braizer and Shannon [1987] presented a knowledge-based modeling system, SIMTOOL, for automatic programming of AGV system simulation models. It was an automatic programming system, written in Turbo-Prolog, that generates computer code in the SIMAN language for the required model. Vehicle parameters defined by the user were number of vehicles, empty and loaded vehicle speeds, initial vehicle positions, processing times, AGV track zone length, etc.

Gong and McGinnis [1990] described a simulation code generator (SCG) that converts input data from a designer into a SIMAN program for evaluating an AGV system with unidirectional guide path network. The SCG compiled the

description file into a simulation code which then was itself compiled into executable code. The input data were in two categories: geometric data and non-geometric data. The geometric data consisted of the guide path network and location of P/D points. The non-geometric data included vehicle specifications, workstation characteristics, part process routes, etc. In their model, a process-oriented approach was applied in which parts and vehicles were treated as entities and machines and buffers were treated as resources.

2.2.2.3.2. An Object-Oriented AGV Simulation Study

In general, the above simulation programs specific for AGV systems are flexible and reduce the complexity of the task of simulating AGV systems. However, they do not offer flexibility, extensibility and reusability.

King and Kim [1995] developed AgvTalk, an object-oriented simulation tool for the design and analysis of AGV system configuration and control. The model was composed of an AGV system, a material handling system, production system, and an interface through which the material handling system and the production system communicated with each other.

AgvTalk includes 25 object classes and more than 300 object methods in its library. Smalltalk-80 has been used as the programming language. Window-based user interface in AgvTalk is supported by MVC (Model-View-Controller) triad in Smalltalk-80 [Lalande and Pugh, 1990]. Defining and changing the system requirements and specifications is done by using Graphical User Interface in AgvTalk. The system initialization includes layout model specification, entity flow specification, and control model specification. In the layout model specification, the control points, distances, zones, and uni- or bi-directional path sections are defined.

In the entity flow specification, vehicle information, workstation information, part and buffer information are included. The control model

specification consists of the vehicle initiated and workstation initiated dispatching rules.

2.3. Conclusion

Simulation is widely used to evaluate the system performance for both real and proposed AGV systems. Many attempts have been made for developing AGV simulators either for specific problems or as generic simulators. There is a strong need for a generic and extensible AGV simulator with broad capabilities. Review indicates that such a simulation system is not currently available at Virginia Polytechnic Institute and State University. Hence, the purpose of this research is to create this simulation system.

Chapter III

REQUIREMENTS FROM A GENERIC AGV SIMULATOR

3.1 Introduction

A generic AGV simulator must provide many features that are needed for simulating a variety of AGV systems. The following sections describe the possible features of a generic AGV simulator.

3.2 Performance Measures

Various performance measures may be required from a generic AGV simulator. These performance measures may be categorized as AGV system statistics, workstation statistics, and job statistics. AGV system statistics consists of vehicle statistics and zone statistics.

Vehicles can be in several states during the simulation. These states are idle, assigned to a job and empty, occupied by a job and loaded, charging, failed, and blocked. The idle status of a vehicle is also in different categories. An idle vehicle may be parking, moving, or waiting at a specific location. Vehicle statistics must include the total time spent in each of these states and their percentage to the total simulation time. In addition, if there are multiple vehicle types, the percentage of total time spent in each status must be categorized for each vehicle type, separately.

Another vehicle statistic is the vehicle response time to calls or job requests. The vehicle response time is equal to the sum of job waiting time until a vehicle is assigned to the job and empty vehicle traveling time to the requesting station. The job waiting time is the time between the initial job request and the assignment of a vehicle to the job. If an idle vehicle is assigned to a job without

a delay, the job waiting time is zero. On the other hand, if the job waits for some time before the assignment, the job waiting time is greater than zero and is a part of the vehicle response time. Distributions for total vehicle response time, job waiting time, and vehicle traveling time may be provided for each vehicle type in the vehicle statistics.

Information regarding the number of loads carried by vehicles may also be required from the simulation model. A distribution for the number of loads must be provided for each vehicle type in the system. These data may also be collected for priority vs. non-priority job types separately.

Zone statistics that can be provided in the simulation package include maximum length of time spent in a zone, distribution of waiting times in queue, average waiting time in queue, maximum blocking time occurred in the overall system by zone name and vehicle type, maximum queue length, total number of vehicle visits to a zone, and the number of vehicle visits in which a blockage occurred. At intersections, the zone statistics may be kept separately for each direction.

Workstation statistics consists of machine statistics and buffer statistics. Machines may be in several states such as idle, operating, being setup, broken, or blocked. Blockage of a machine occurs when a machine finishes processing and can not proceed to the next job because of buffer limits and/or absence of a vehicle. Machine statistics must include the percentage of time spent in each of the machine states.

Buffers are classified as input buffers and output buffers. A buffer may be empty, occupied and not blocked, or blocked. Blockage of an input buffer is caused by the machine in front of it. On the other hand, blockage of an output buffer is caused by the absence of a vehicle. The percentage of time spent in each of the buffer states may be provided in the buffer statistics.

Jobs may be in a number of states: waiting to be processed, being processed, being setup, waiting for transport, and being transported. The time spent in each of these states can be collected into job status histograms for each job type, separately. It is also possible to keep statistics of all states for all jobs entering the system. However, this requires a huge computer memory capacity. Other job statistics that can be covered in the simulation package include total time spent in the system for every job type and in the form of histograms, total number of jobs completed from each job type, and maximum number of jobs waiting for an idle vehicle at a specific workstation.

Although this is not the entire list of all possible statistics that may be needed for the evaluation of AGV systems, all basic performance measures that have been used by the past researchers can be found in the above statistics options. More importantly, a generic AGV simulator must be extensible in terms of adding new performance measures.

3.3. Production Management and Job Request Types

Job arrivals to the system may be programmed in two ways: 1) The orders from each job type may be given by the user at certain arrival times at the beginning of the simulation. 2) Time between job arrivals may be specified at certain workstations in order to trigger new orders from these workstations.

In the former, the job is routed from one workstation to another according to the operations sequence. Every workstation provides a list of available operations so that jobs can be routed to the right workstations. For the latter, the operations sequence of job types may or may not be specified. If specified, a job arrival triggers the first operation in the sequence, and then, other operations are read from the operations sequence table. If not specified, a job arrival triggers only one transport from the origin of the request to the specified destination.

In this case, transport request may come either from the origin station or from the destination station. The time between job arrivals can be either constant or randomly distributed.

In flexible manufacturing systems (FMS) applications, one operation may be done at more than one workstations. In such cases, alternative routing of jobs is possible. A generic AGV simulator must include a modular router function to specify the method to be used for the routing strategies. The user may change this function according to specific rules applied for the routing of jobs in a specific AGV system. For example, a simple router function can be programmed so a job is not assigned to a workstation if the workstation is not available (broken, or full) at a particular time.

3.4. Independent Workstations

Some workstations in a manufacturing system may not be interacting with the AGV path and the vehicles. However, these workstations may be interacting with other workstations which load/unload parts from/to AGVs through pickup and deposit locations.

For example, operation sequence of a part is specified as: enter-mill-turn-face-exit. The turning operation is done in a workstation which has no connection to the AGV system. AGVs unload the part at milling machine and after the milling operation, the part is transferred to the turning machine without using an AGV. Similarly, it is transferred to the facing machine without interacting with the AGV system. After the facing operation, an AGV picks up the part and carries it to the exit point. Here, the turning operation is done at an “independent workstation.” In other words, this operation is independent from the AGV system. The AGV simulator must have the capability of including such independent operations by considering their machining times. In the example model, the time of arrival to the facing station depends upon the machining time at the turning station.

3.5. Assembly Operations

In an assembly system, parts coming from different workstations may be attached together and become a single part after an assembly operation. For example, parts A and B are transported to an assembly cell after which they become a single part. A single transport request is triggered after the assembly operation. Here, assembly charts and detailed operation process charts are needed for simulating such an assembly system. These charts may not be provided as an input to the simulation package. However, assembly systems may still be simulated without considering the actual parts flow in the system. It is possible to concentrate on the unit loads carried by AGVs without knowing the contents of the unit load. In this case, the only necessary information is the origin and the destination of the loads and the machining times and loading/unloading times.

3.6. Machining Times

Processing times and setup times at workstations can be either machine specific or part specific. The generic AGV simulator must provide both options to the user. If the machining times are part specific, the processing time and the setup time for each part type must be input as a part information. If the machining times are machine specific, the processing time and the setup time for each machine must be specified as a characteristics of the machine. In both types, the time definitions may be either constant or randomly distributed.

3.7. Buffers and Interactions

The simulator must allow both limited and unlimited buffers in the system. For limited buffers, if the buffer is full, a vehicle will not be able to unload part(s) onto the buffer. In this case, the vehicle may wait at that location until the buffer accepts new loads. While it is waiting, the other vehicles may be blocked because of the waiting vehicle. However, if a spur (a dedicated path section)

exists for loading/unloading parts at a workstation, the waiting vehicle will not block other vehicles. The generic AGV simulator must utilize stations both with a spur or without a spur.

The waiting times at buffers must be considered in the collection of statistics. In real life, a buffer may be a conveyor, a container, or maybe a turning index. The transfer time between the vehicle and the buffer, and the loading/unloading time between the buffer and the machine must also be considered in the simulation. In some cases, there is not a buffer associated with a workstation. Moreover, a machine may have a capacity of processing only one unit at a time. The generic AGV simulator must be able to simulate a workstation without buffers or with a total capacity of one unit.

There are three types of interactions between a workstation and a vehicle. These are pickup and deposit interactions at the same place, only pickup interaction, and only deposit interaction. The pickup and deposit interactions at the same place can be in two different types. First, both pickup and deposit interactions can be done at the same time without interrupting each other. Second, only a pickup or a deposit interaction can be done at one time, but not at the same time. The simulator must utilize all types of interactions between workstations and vehicles.

3.8. Breakdowns for Machines and AGVs

Both machines and AGVs may have breakdowns. The simulator must provide breakdowns at given time intervals and for specified duration. Time interval between two consecutive breakdowns and repair times for each vehicle and for each machine must be defined. The time values may be either constant or randomly distributed. The user may also wish to activate/deactivate breakdowns for a vehicle or for a machine during the simulation.

There may be two types of vehicle breakdowns. The first type is a breakdown in which the vehicle is taken to a maintenance area. The second type is a breakdown in which the vehicle stays at its current location and waits for a quick repairing operation. The second type may also be considered as minor delays due to the interruptions on the way of the vehicle. The time for recovering the vehicle is different in each type of breakdown. The simulator must provide both breakdown types and the percentage of their occurrences must be given by the user.

In addition, when a vehicle fails, its current load may or may not be taken away from it. First alternative is to carry the loads to their destinations via other material handling sources. Another alternative is to leave them on the vehicle until it is repaired.

3.9. Dispatching Rules

If a vehicle becomes idle and there are more than one workstation requesting a vehicle, the vehicle selects to serve one of those stations according to a strategy called vehicle-initiated dispatching rule. Some vehicle-initiated dispatching rules are longest waiting time, shortest distance, and randomized choice.

If a station makes a transport request and there are more than one idle vehicle, the station makes a selection among the idle vehicles according to a strategy called workstation-initiated dispatching rule. Some workstation-initiated dispatching rules include longest waiting time, shortest waiting time, shortest distance, randomized choice, minimum utilization, maximum velocity, and, maximum battery charge.

A generic AGV simulator must provide these dispatching rules. User-defined dispatching rules may be added to the current options, as well. Furthermore, the simulation may be run without using any of the dispatching rules. In this case, the user must be able to turn all the dispatching rule settings off.

3.10. Battery Charging

Battery charging feature may be included in a simulation model. Some AGV systems employ the opportunity charging technique. For this reason, an opportunity charging option which can be activated upon the user's request must be provided. It is also possible to exchange batteries with spare batteries which have already been charged. The time spent for changing the batteries is obviously less than the time spent for charging the batteries. An AGV system may even employ different battery charging/changing techniques for different vehicle types. This must be considered in the simulation of the system.

Simulating the battery charging feature needs information related to the characteristics of the batteries. A generic AGV simulator must utilize additional data for the battery charging feature. These data include maximum battery charge capacity, battery consumption rate, charging current, battery safety level, and initial charge level of the battery.

If a vehicle needs to be charged when it is loaded, its load must be carried to its destination either by the vehicle itself or by other means of material handling. If a vehicle is assigned to a job and traveling towards that job, in case of low battery level, the job is assigned to another vehicle.

3.11. Vehicle Routing

There are two types of vehicle routing strategies in an AGV system: fixed routing and dynamic routing. In the fixed routing, vehicle routes are fixed. These fixed routes are set at the initialization stage and they are not changed during the execution of the simulation model. They may be either specified by the user or calculated by the computer program so that the optimal traveling distances between locations are obtained.

In the dynamic vehicle routing, vehicle routes are changed dynamically at decision points. A decision point is a control point where alternative vehicle routing is possible. A modular router function may be defined for rerouting vehicles at decision points. Rerouting may be necessary if there is a blockage on the current path of a vehicle. The modular router function gets two parameters from the AGV system: the list of the currently blocked zones and the list of the destination sequence of a vehicle. The former contains the names of the zones which are currently blocked. The latter contains all the zones which will be visited by a vehicle until the vehicle reaches to its destination.

If one of those zones is blocked, the vehicle may be rerouted at the next decision point. For dynamic vehicle routing, the generic AGV simulator must provide these two parameters for the execution of the modular router function.

3.12 Bi-directional Flow Logic and Multiple Loads

In many AGV systems, a small section of the AGV path is dedicated for two-way traffic and the rest of the path is uni-directional. For this reason, the AGV simulator must allow the user to define any individual section as being bi-directional. Even a single zone can be bi-directional while the rest of the path is uni-directional.

In the simulation, AGVs must be capable of carrying multiple loads at the same time. For example, if towing vehicles are used in an AGV system, the corresponding vehicles in the simulation model must be able to carry multiple loads, as well.

3.13. Priorities and Vehicle Assignments

In an AGV system, the vehicles may be programmed to answer a rush order first. Some trips may have higher priority than others. For example, a trip from storage to production area is considered more important than a trip from production area to the storage. It is, therefore, necessary to distinguish jobs according to their priorities. In addition, vehicles may have priorities, too. For example, if an empty vehicle and a loaded vehicle come to an intersection at the same time, the loaded vehicle may have the priority.

In some AGV applications, there are multiple types of AGVs that are used for different tasks. The user may wish to assign vehicles to certain pickup or deposit locations. It is also possible to assign vehicles to certain job types. This may be necessary due to the variety of load size capacities of different vehicle types. The simulator must provide these vehicle assignments, but as default, all vehicles must be able to serve to all job types and all workstations.

3.14. Simulating Wireless AGVs

In wireless AGV applications, there is a virtual path which the vehicles follow. A virtual path is different from a wired path only in terms of their control systems and expanding capabilities. The conceptual path exists in both types. Therefore, a simulator which uses the control point concept, may also simulate wireless AGV systems. The modeling logic is not affected when the path is virtual.

3.15. Traffic Management

Blocking occurs when two or more AGVs try to use the same section of guideway at the same time. There are three possible types of conflicts in an AGV system. The first type of conflict occurs when two vehicles travel in the opposite directions and move into the same zone at the same time. This is only possible in a bi-directional flow pattern. If this type of conflict is controlled by a traffic controller in the real AGV system, the simulation model must also eliminate such a conflict. The second type of conflict occurs when a faster moving vehicle collides with a slower moving vehicle in front of it. In the zone controlling technique, this type of conflict is eliminated by stopping vehicles at the zone control points. In the forward sensing technique, there is a specific distance between vehicles in order to prevent vehicles from colliding. Therefore, the simulator must not allow this type of conflict when the model is representing any of these controlling techniques.

The third type of conflict occurs at intersections. A vehicle spends a certain time while it is passing an intersection. During this time interval, if other vehicles come to the intersection, they must wait until the area becomes clear. Normally, first arriving vehicle will occupy the intersection area first. However, there may be some specific traffic rules that are applied at intersections. For example, if two vehicles are approaching to an intersection and one of them has a high priority order, the right to pass may be given to this vehicle, even if it is the second vehicle reaching the intersection. Similarly, a loaded vehicle may have the priority against an empty vehicle. The simulator must provide these rules and it must be possible to add new traffic rules, if desired.

3.16. Animation

By using the animation capability, vehicle positions and the status of each vehicle (idle, assigned, etc.) are visible during the simulation. Since the computer time required for a simulation run is longer when animation feature is activated, the animation feature must be provided as optional and the user must

be able to turn it off at any time. Different vehicle types and different job types may be presented with different pictures in the animation. Machine states, such as operational, broken, idle, etc., may also be presented by different pictures.

3.17. Layout Drawing

Most of the existing generic AGV simulators represent the AGV layout as an input from a formatted ASCII file. Some simulation studies used the method of transferring the layout data from a CAD drawing. One standard output from a CAD drawing is a **.dxf** file and it can be used as input to the simulation program.

In ideal case, the CAD drawing contains all necessary information for the simulation. Basic data to be extracted from the drawing include point types (pickup interaction, deposit interaction, control point, etc.), and point coordinates. Complex data that may be extracted from the CAD drawing include direction of flow and the possibility of bi-directional flow at each section.

Machine related layout data, such as exact locations of machines and buffers, are ,generally, not shown in the CAD drawings of AGV systems. Instead, only the interaction points between machines and vehicles (e.g. pickup, deposit points) are provided.

3.18. Conclusion

Although simulating a typical AGV system may not need all these features, it is necessary to include most of them in a generic simulation package. However, some features, such as simulating assembly operations, may be optional and can be added in the future. The proposed AGV simulator will be able to simulate a wide range of AGV systems, but not all AGV systems. Table 3.1. compares SLAM, SIMPLE++, AgvSim, AgvTalk and the proposed simulator in terms of their capabilities.

Table 3.1. Comparison of existing simulation programs for AGV systems

Capabilities	<i>SIMPLE</i>	<i>AgvSim</i>	<i>AgvTalk</i>	<i>SLAM</i>	<i>PROPOSED</i>
Battery Charging	X	X	X		X
Breakdowns		X	X	X	X
Multiple vehicle types		X	X	X	X
Priority assignment				X	X
Vehicle assignment					X
Layout drawing					X
Animation	X				X
Job request types		X			X
Independent stations					X
Assembly operations					
Dispatching rules	X	X	X	X	X
Vehicle routing		X	X		X
Bi-directional flow		X	X		X

Chapter IV

MODEL DEVELOPMENT

4.1. Introduction

The simulation tool developed in this research utilizes the AGV library of the commercial package SIMPLE++. The modifications made to the SIMPLE++ AGV library will be explained in this chapter. Some terms have been taken from the SIMPLE++ terminology. Therefore, it is best to refer to the SIMPLE++ User's Manual if a term is not known. In this chapter, first, the existing AGV objects library of SIMPLE++ will be introduced. Second, the modifications made to this library will be explained in detail. Revisions include both modifying the existing objects of SIMPLE++ AGV library and adding new objects to this library. The modification of the existing objects is classified as changes related to AGV system and changes related to production system. Changes related to AGV system affect the existing AGV system related objects such as AGV, AGV system manager, and path elements. Changes related to production system affect the existing production system related objects such as Part and Production Manager. The new AGV application objects developed in this study include AGV Model Frame, Statistics, Route Manager, Maintenance Station, Battery Charge Station, and Model Builder.

4.2. Object-Oriented Modeling Approach for AGV Systems

An AGV system consists of physical objects such as vehicles, jobs, AGV system controller, workstations, buffers, load transfer mechanisms, charging stations, maintenance stations, and zones. These physical objects can be represented as instances of simulated objects by using the object-oriented modeling approach.

In an object-oriented program, messages passing between the objects correspond to the function calls in the conventional programming. For example, in a centralized AGV system, vehicles may get their destinations from the central controller. When a vehicle needs to receive information about its destination, it sends a message to the controller object and then updates its status according to the response from the controller. The inheritance feature of object-oriented programming allows the construction of new objects by modifying the characteristics of the existing objects. For example, a number of vehicle subclasses representing different vehicle types may be produced from a general vehicle class. The general vehicle class defines the general characteristics of a vehicle object such as speed, load capacity, battery consumption rate, maximum battery capacity, etc., while each subclass defines its own specific characteristics.

4.3. SIMPLE++ AGV Application Objects

SIMPLE++ is an object-oriented modeling environment. Everything in SIMPLE++ is done by using “basic objects” and the SimTalk language. Objects are visible icons in an objects template, or library. The basic objects are the skeleton of SIMPLE++ and are classified as “physical objects” and “information objects.” Basic physical objects are Transporter, Entity, Line (e.g. conveyor), Processor, Track (e.g. path), Container, and Storage. Basic information objects are Method, Generator, List, Table, Display, and Comment. New objects can be constructed by interactively combining these basic objects. These new objects are called “application objects.” SIMPLE++ has an application objects library (SIMPLE++_AGVS) specifically developed for AGV systems (see Figure 4.1).

Application objects defined in this library include agvs_manager (AGV system manager), ProdMgr (production manager), station (workstation), AGV, Part, agvs_station (parking station), agvs_dock_station (dock station), agvs_straight (straight line), agvs_bend (curve), agvs_junction, agvs_branch, and agvs_tl_br (terminal line branch).

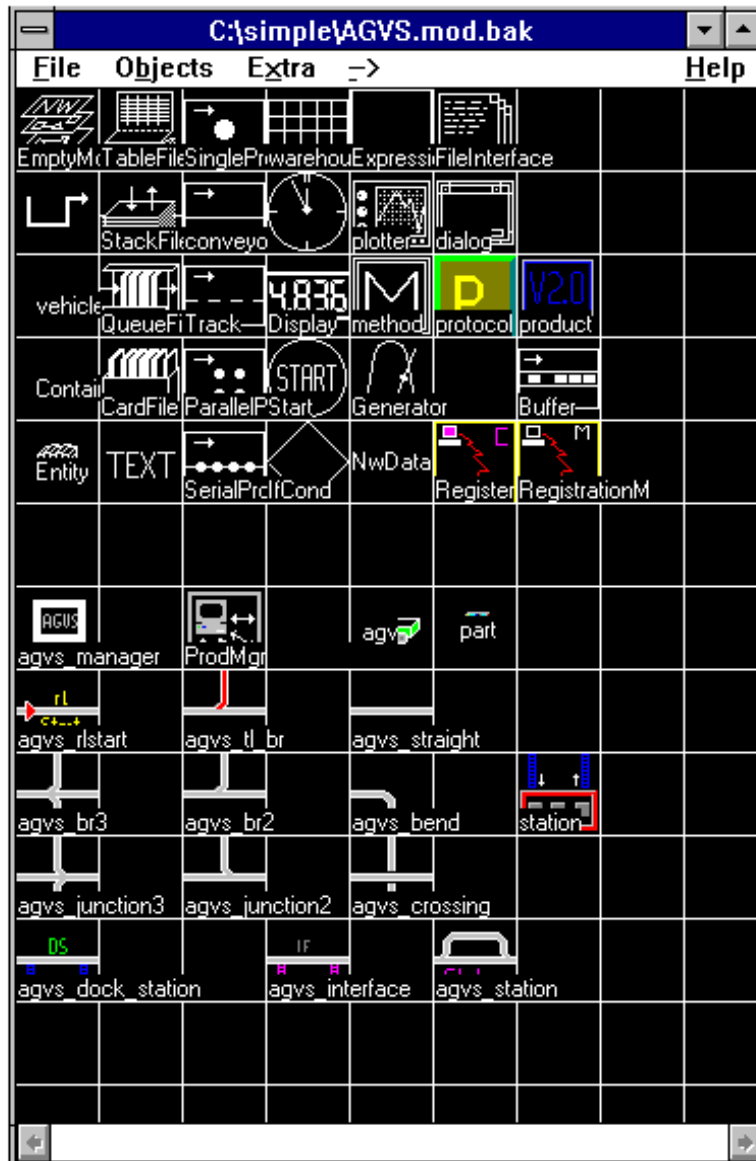


Figure 4.1. The SIMPLE++ AGV objects library

In this research study, the existing AGV application objects library have been expanded in order to build a generic AGV simulation environment. This enhancement includes both modifying the current objects and adding new application objects to the current library. In the following sections, the existing application objects will be introduced and the enhancements made in this study will be explained.

4.4. Existing AGV Application Objects

SIMPLE++_AGVS is a collection of application objects that is used to model AGV systems. The following sections will describe the functions of these application objects. For each application object, a dialog window is provided for changing the specific attributes of the object. In every AGV simulation model, at least three objects must be included. These objects are agvs_manager, ProdMgr, and EventController.

4.4.1. agvs_manager (AGV System Manager)

The agvs_manager is an application object that is used for managing the vehicles and the zones in the AGV system. All vehicles are registered to the agvs_manager. The number of vehicles, simulation reset alternatives, and control rules (e.g. vehicle-oriented dispatching rules, workstation-initiated dispatching rules, etc.) are set in this object. At the end of a simulation, the user may either delete all current vehicle specifications or keep them for the next simulations. Similarly, distance registrations between pickup/deposit points may be reset or kept unchanged from one simulation to another. These are possible reset options provided in the SIMPLE++_AGVS library. By opening the dialog window of the agvs_manager, desired reset options may be selected.

Whenever a vehicle changes its status, the agvs_manager updates tables showing the current vehicle states and job assignments. Upon receiving a transport request, the agvs_manager selects a suitable vehicle. If there is not an available vehicle, the job is placed into a queue file in the agvs_manager. As soon as a vehicle becomes idle, it is assigned to one of the waiting transport requests according to the specified vehicle-initiated dispatching rule.

4.4.2. Production Manager (ProdMgr)

The ProdMgr is an application object that is responsible for the production planning of a manufacturing system. Workstation and job types information are stored in the ProdMgr. All workstations and job types are registered to and managed by the ProdMgr. Workstation information includes names of workstations, types of workstations, and operations available at each workstation. Job types information includes the operations plan, machining times (if part specific), and the number of orders for each job type.

If machining times are part specific, the processing times and the setup times for each part type are defined in an operations plan table which is contained in the ProdMgr. In the current model, these machining times can be input as constant values, but not as randomly distributed values.

The dialog window of ProdMgr has a button known as Orders, which is used to set the order quantities, due dates, and release dates for each job entering the system. These settings are performed at the beginning of the simulation. A job is started at the specific release date and routed according to its operations plan. After completion of each process, the next operation of the job is received from the ProdMgr object. All information regarding the operations plan are read from the tables in the ProdMgr.

The ProdMgr also monitors the availability of each workstation, at any time, with respect to their capacities. For example, if the number of jobs in a

workstation has reached its total capacity and the workstation can not accept any more jobs for processing, the station is assigned as not available. In case of alternative routing, there will be more than one possible workstation that can be addressed for a specific operation. If one of them is not available, that workstation will not be considered for a job assignment.

4.4.3. Simulation Clock (EventController)

The EventController is a basic object used for managing and controlling the execution of a simulation and is needed for all simulation models. It includes buttons such as Start, Stop, Init, and Reset. By pressing the Init button, all methods with the name Init in every object in the simulation model are performed. This button is used for initializing the simulation parameters. Similarly, the Reset button calls the methods with the name Reset in every object and stops the simulation clock. The current simulation time is displayed on the dialog window of the EventController. More information can be found in the SIMPLE++ User's Manual.

4.4.4. Workstation (station)

The station is an application object which represents a physical workstation. It can be any type of machine (mill, drill, etc.) or even a worker. In a station object, jobs are processed one by one. In other words, the workstation can process one unit at a time. A station object may interact with AGVs via a dock station. The station consists of an entry place (input buffer), a work place, a setup place, and an exit place (output buffer).

After completion of a process, a job is transferred to the corresponding dock station if the current workstation is not the last destination of the job. At the

dock station, the job waits for an AGV. If the current workstation is the last destination, the job is terminated after the collection of statistics.

During the construction of a station object, the operations that can be performed by the workstation must be defined. These operations match with the operations listed in the operations plan table in the ProdMgr. The dialog window for a station object can be used to set the machining times for each section of the station object (Entry, Setup, Work, Exit). The entry place and the exit place of a station represent the input and output buffers at the workstation (e.g. a conveyor or a container). Moreover, workstation breakdowns may be activated by specifying the duration of a breakdown and the time interval between two consecutive breakdowns. These time values can be either constant or randomly distributed.

4.4.5. AGV

AGV is an application object that has been derived from the basic object Transporter. When creating a new AGV, its speed, battery specifications, current destination, load size, driving direction (forward or backward), and the current status (free, assigned etc.) must be initialized. Unless the user modifies these settings, the model gets the default values from the agvs_manager.

The default driving direction of an AGV is forward. In the SIMPLE++_AGVS library, the vehicles are allowed to only move forward. One exception is at the terminal lines. At a terminal line, the vehicles have to move backwards. Other than this exception, the vehicles move only at one direction on the path. Bi-directional flow logic has not been included in the current AGV library.

The default load size of an AGV is one unit. Methods used at a dock station for loading/unloading AGVs have been arranged so that only one unit can be transferred from/to the AGV at a particular time. Therefore, the current AGV

library does not allow the multiple-load carrying capability. Current status of a vehicle can be seen in the dialog window of an AGV. This status can be free, assigned, or occupied. If it is set to free, the vehicle can be assigned to a new order. When the vehicle is empty and traveling towards an order, it is in the assigned status. When loaded, it becomes occupied.

At parking stations, vehicles may be placed in a queue. An attribute called `vd_property` is used for the control of AGVs at parking stations. If it is equal to "inQueue," the vehicle is in the parking station, but not in the first position. Hence, it may not be assigned to an order. Battery related attributes of the AGV object are current charge of the battery, battery consumption rate, charging current, battery safety level, and the name of the method to be called when the battery charge of an AGV reaches to the specified safety level.

During the initialization, vehicles may be placed either at a parking station or at a path segment. If a parking station exists in the model, vehicles are automatically located into the parking station. Otherwise, the vehicles are located into arbitrarily selected path sections.

4.4.6. Part

Part is an application object which is derived from the basic object Entity. It represents any kind of job which is transported and processed in the system. In the `SIMPLE++_AGVS` library, a part gets its next destination from the `ProdMgr` object according to the operations plan defined for its job type. Part objects interact with AGVs, stations, and the `ProdMgr` object, but not with the AGV path objects. It is always carried on a vehicle. When it is on a vehicle, the destination of the vehicle is set equal to the current destination of the part.

4.4.7. Path Elements

Path elements are objects where the AGVs can move. The following sections will describe the functionality of each path element provided in the SIMPLE++_AGVS library. Some of these path elements have similar functionality and will be described once.

4.4.7.1. agvs_straight and agvs_bend

The agvs_straight represents a straight path section which can be defined as a vertical or a horizontal line. Similarly, the agvs_bend is a curve element which can be turned to fit into a specific shape. When a vehicle comes to the end of an agvs_straight or an agvs_bend, it is sent to the successor of that path element. The length of a path object can be set from its dialog window.

4.4.7.2. agvs_br2 and agvs_br3

At a branch, an AGV decides which direction it should move. The agvs_br2 and agvs_br3 are application objects that are used for modeling 2-way and 3-way branches, respectively. The assignments for the directions and the information about reachable destinations are stored in the sign lists that have been defined in these application objects. An entering AGV searches the sign lists to find which direction it must take to reach its current destination. If the destination is found in one of the sign lists, the AGV is moved to the corresponding direction. These sign lists are created based upon the shortest distances between branches and the destination points.

4.4.7.3. agvs_junction2 and agvs_junction3

Similar to the branches, junctions on an AGV path can be modeled by using the junction elements `agvs_junction2` and `agvs_junction3` for 2-way and 3-way junctions, respectively. Since, bi-directional flow logic is not utilized in the SIMPLE++_AGVS library, the vehicles are sent to the default successor of the junction element. In other words, no traffic decision is made at a junction. Furthermore, the first incoming vehicle to the junction has passing priority. There are not optional traffic rules provided in the current AGV library.

4.4.7.4. AGV Parking Station (`agvs_station`)

AGV parking stations are modeled by using the application object `agvs_station`. The maximum number of vehicles that can be located in the parking station must be set from the dialog window of the `agvs_station` object. The number of AGVs in the system can be greater than the total capacity of the parking station. AGVs in the parking station are placed in a queue. Only the first vehicle in the queue can be called for a transport order.

When a vehicle enters an `agvs_station` object, if its current destination is the `agvs_station` object, it is held in a waiting place. Later, a transport request causes the vehicle to come back to the main path from this waiting place. If the current destination of the vehicle is not the `agvs_station` object, it is sent to the next path section.

4.4.7.5. Register-line Start (`agvs_rlstart`)

Each path element contains a table, called `rl_dist`, which contains the registered distances between a path element and the branches on the path. These tables are used to calculate the distances between path elements. They are initialized by sequentially passing register-line messages through the whole system during the construction of the model. An AGV located on an `agvs_straight` object will calculate its distance from a destination point by adding the distance between the `agvs_straight` object and the closest branch (via the

rl_dist table) and the distance between the branch and the destination object. Therefore, in the SIMPLE++_AGVS library, distance between the current location of an AGV and a reachable destination is calculated only if there are branches in the system. If there is not a branch object in the model, an agvs_rlstart object must be replaced with one of the agvs_straight objects in the model. The agvs_rlstart and the agvs_straight are similar objects in terms of their functions.

4.4.7.6. Terminal Line Branch (agvs_tl_br)

The agvs_tl_br is a branch object which is used as an entrance to a terminal line. A terminal line is a dead-end path section and must be connected to the main path via an agvs_tl_br object. When an AGV enters an agvs_tl_br object and if the current destination of the AGV is on the terminal line, it is directed towards the terminal line. Otherwise, it is sent straight on without entering the terminal line.

The maximum number of vehicles that can occupy the terminal line must be set from the dialog window of the agvs_tl_br object. If, at any time, the current number of vehicles in the terminal line reaches the maximum value, incoming vehicles will not be allowed to enter the line.

4.4.7.7. Dock Station (agvs_dock_station)

A dock station represents a reachable destination point on the AGV path. It is a simple path element, not a workstation, which is used for the interaction between AGVs and workstations.

An agvs_dock_station object must be attached to a station object which represents the actual destination for a Part object. When an AGV enters into an agvs_dock_station object and if the current destination of the AGV is the agvs_dock_station object, the AGV is stopped, loaded or unloaded, and then

moved to the successor of the `agvs_dock_station` object. If the destination is somewhere else, the AGV is not stopped.

Parts coming to an `agvs_dock_station` object may or may not be directly transferred to the station object. If there is a transfer delay between the time of unloading parts from the AGV and the time of arrival of the parts into the workstation, this delay can be modeled in the `agvs_dock_station` object.

4.5. Revisions to the Existing AGV Application Objects

The existing application objects defined in the `SIMPLE++_AGVS` library have been modified to meet the requirements of the generic AGV simulator. These modifications can be classified as AGV system-related changes and production system-related changes.

4.5.1. Changes Related to AGV System

AGV system-related application objects include `AGV`, `agvs_manager`, and path elements (`agvs_straight`, `agvs_bend`, etc.) The required features and capabilities described in Chapter 3 have been developed by altering the structure of these application objects. In this section, the new features added to the AGV system-related application objects will be listed and the results of this research study will be outlined.

4.5.1.1. Vehicle Types

The `AGV` object in the current `SIMPLE++_AGVS` library does not have an attribute for describing the vehicle type. All vehicles in the model are shown with

the same picture. An attribute, named `vehicleType`, has been defined as a custom attribute (refer to the SIMPLE++ User's Manual) of the AGV object in order to distinguish vehicle types from each other. By altering the `Init` method of the `agvs_manager`, it has been made possible to assign vehicle types (e.g. towing and unit-load) to AGVs. For each of these vehicle types, a default picture has been drawn. The user must draw additional pictures to define new vehicle types.

4.5.1.2. Vehicle Assignments

There are several ways of vehicle assignments. First, an AGV can be assigned to serve only for a specific workstation. When the AGV becomes free, it is directly sent to the specific workstation and waits there even if there is not a transport request. An attribute, called `IdleVehicleStatus`, has been added to the AGV object in order to accomplish this kind of vehicle assignment. If the `IdleVehicleStatus` is set to "WaitAtMachine," the vehicle will serve only for the specified workstation.

Second, vehicles may be assigned to serve only for specific workstations, but not for others. The difference from the first type of vehicle assignment is that, now, the vehicle can serve more than one workstation. In fact, this assignment type is the general case, while the first type is a specific case. A list containing the suitable vehicles for each workstation has been defined so that the `agvs_manager` will be able to distinguish which vehicles can serve for which workstations. The user may alter the suitable vehicles list to assign certain vehicles to a specific workstation. This can be done by using the dialog window of an `agvs_dock_station` object.

By pressing the vehicle assign button in the dialog window, a list of vehicles appears. If a vehicle is deleted from this list by the user, that vehicle will not serve for the `agvs_dock_station` object to which the dialog window belongs.

Third, vehicles may be assigned to serve for specific job types in the system. Job type and the corresponding operations plan are input from the dialog window of the ProdMgr. The suitable vehicles may also be restricted with respect to the job types. Similar to the second case, all vehicles are assigned to all job types, as default. If a vehicle is erased from the suitable vehicles list of a job type, the vehicle does not serve for that specific job type.

To accomplish the above vehicle assignments, job assignment methods that are defined in the agvs_manager object have been modified and a table which shows the available vehicles lists has been added to the dialog window of the ProdMgr object.

4.5.1.3. Vehicle States

Three vehicle states have been defined in the current SIMPLE++_AGVS library: free, assigned, and occupied. These vehicle states were explained in section 4.4.5. Three additional vehicle states have been defined in this study. They are failed, blocked, and charging. In addition, the free status has been divided into three pieces: parking, moving, and atMachine. An idle vehicle may be parking at an AGV station, moving empty on the AGV path, or waiting at a specific workstation (the atMachine status). The moving empty status may occur when an AGV finishes unloading and moves towards a parking station. The waiting idle (atMachine) status may occur if the AGV is assigned to serve for a specific workstation.

An AGV is in the failed status when a breakdown occurs. There may be several reasons for a vehicle breakdown. Some of the reasons include bumper contact, on-board controller failure, empty battery, and worker interruption. During the breakdown, vehicles change their colors in the animation. Vehicle breakdowns will be explained in detail in the next section. When an AGV is sent to a battery charging station, its status becomes charging. Vehicle blockage may occur due to traffic congestion and/or vehicle breakdowns. An AGV is in the

blocked status when it can not move while it should move. A blocked AGV blocks other vehicles which are directed to the same path section.

4.5.1.4. Vehicle Breakdowns

The SIMPLE++_AGVS library provides breakdown methods for workstations, but not for AGVs. A Generator object and a corresponding method have been built for setting vehicle breakdowns at certain time intervals and for a specific duration. The time values may be either constant or selected from random distributions. A switch for activating/deactivating vehicle breakdowns has also been added to the dialog window of the AGV object.

Some breakdowns need the AGV to be taken out from the main path to a maintenance area, while some other errors may be recovered right on the path. In general, the latter takes much less time than the former. Therefore, vehicle breakdowns have been partitioned into two types: type A breakdown (the former) and type B breakdown (the latter). If both breakdown types are desired in the simulation, a percentage of the occurrence of each type must be determined by the user.

When an AGV fails, its current load(s) may be carried to their destination(s) via other means of material handling or they can be left on the failed AGV. A Boolean attribute has been added to the AGV object in order to specify whether the load is transferred from a failed AGV or not.

For some breakdowns, the current load of the AGV may be kept on the AGV, while for some other breakdowns, they are transferred to their destinations. Therefore, another custom attribute has been defined for determining the percentage of the frequency of both cases.

4.5.1.5. Shift Work for AGVs

Some AGV systems utilize vehicles that are used from shift to shift, separately. In such systems, while some vehicles are operating in the production system, other vehicles are charged at a battery charging station. They change shifts like the workers in a factory. To be able to simulate this type of systems, a Boolean attribute, called `shiftWork`, has been added to the AGV object. If the Boolean attribute is true, vehicles work only during the specified time intervals. At other times, they are parked and/or charged.

4.5.1.6. Pickup/Deposit Interactions

There are three types of interactions between vehicles and workstations: pickup and deposit at the same place (P/D type), pickup only (P type), and deposit only (D type). Only the P/D type interaction has been provided in the `SIMPLE++_AGVS` library. An interaction point is represented by an `agvs_dock_station` object. The P type and D type interactions have been made possible by building new methods in the `agvs_dock_station` object. Additionally, two new dock station pictures have been drawn to represent P type and D type dock stations. The P type and D type dock stations are one-way dock stations where the load transfer is only one-way. For instance, at a P type dock station, vehicles are not allowed to unload parts, while at a D type dock station, they are not allowed to load parts.

The picture of a P/D type dock station is shown as a two-way interaction point. During the animation, parts entering and leaving the P/D type dock station can be seen simultaneously. However, a two-way interaction point may be restricted to transfer loads only one way at a time, but not simultaneously. In this case, the picture is shown as a one-way interaction point, but the object acts like a two-way interaction point.

4.5.1.7. Spur Option

Interaction points may have a distinct area (a spur) that is used for loading/unloading parts without blocking the main path. If there is a spur, when an AGV enters the spur, it does not block the main path while it is interacting with the workstation. An AGV system may contain several workstations with or without a spur. The proposed simulator is able to model AGV systems which may have both types of workstations.

An AGV layout drawing is the source for obtaining the spur information. The maximum number of vehicles that can be located simultaneously in a spur is specified in the dialog window of the spur object. The spur object is located in the `agvs_dock_station` object and is similar to a queue where the AGVs wait until their interaction with the workstation ends.

4.5.1.8. Vehicle Speed

AGVs have to slow down at certain sections of the path. The flow speed of vehicles is not only dependent upon the maximum speed capacity of the vehicles, but also it is determined by the current location of the vehicle. For instance, at curves, the safety speed limit for a vehicle must be calculated based on the radius of the curve. Although the modified AGV simulator does not make such speed calculations, it allows the user to specify the speed limit at each zone, separately.

When entering a new zone, an AGV sets its velocity equal to the permitted speed value of the zone. At the end of the zone, the AGV returns to its default (normal) velocity. This feature provides flexibility in setting vehicle speeds at specific zones. By default, all zones allow the vehicles to move with their default velocity.

The speed settings can also be vehicle specific. In this case, vehicle speed changes are based on the status of the vehicle. For example, empty

vehicle speed and loaded vehicle speed may be defined as two separate speed values of an AGV.

4.5.1.9. Multiple-load Carrying and Multiple Destinations

The SIMPLE++_AGVS library has been designed for one unit-load carrying AGVs. Many AGV systems use multiple-load carrying AGVs. Therefore, the SIMPLE++_AGVS library has been modified in order to be able to model multiple-load carrying vehicles.

Two methods, named load and unload, have been developed for executing the loading/unloading of AGVs at the dock stations. Arrival of an AGV triggers one of these methods according to the purpose of the call and the loads are transferred to or from the AGV. For example, the unload method first checks if the first load of the vehicle is destined for the current dock station. If this is true, then the availability of the input buffer is checked. If it is not full, the load is transferred. The same procedure is repeated for all units on the vehicle.

In some cases, AGVs are capable of handling more than one job type at a the same time. In such a case, loads on the AGV may have different destinations. Normally, the closest destination is set as the first destination of the vehicle. In fact, it is best to arrange the total travel distance of the vehicle to be the optimum travel distance.

The proposed simulator first checks the travel distances for all units on the vehicle. The closest destination is selected. By reaching the first destination, all units which are destined for this workstation are unloaded. The next closest destination is then selected and these steps are repeated until all the units on the vehicle are transported.

4.5.1.10. Fixed Vehicle Routing

Vehicle routing methods were categorized as fixed vehicle routing and dynamic vehicle routing. In the fixed routing method, vehicle routes are set once and are not changed during the operation. In other words, vehicles can not make intelligent routing decisions when there is a blockage on their way. The dynamic vehicle routing method is more efficient but harder to control. The SIMPLE++_AGVS library utilizes only the fixed vehicle routing method by employing the sign lists concept.

A sign list is a table which contains the names of possible destination objects that can be reached from an intersection point given a specific direction is selected. For instance, `rl_signL_l` is the name of a sign list which contains the names of possible destination objects that can be reached from the left side of an intersection. Similarly, `rl_signL_r` is the name of a sign list which contains the names of possible destination objects that can be reached from the right side of the intersection. `rl_signL_s` and `rl_signL_f` are the sign lists showing the reachable destination objects on the straight and back sides of the intersection, respectively.

When an AGV enters an intersection object (a branch or a junction), the current destination of the AGV is found in one of the sign lists. The vehicle is moved to the direction to which the selected sign list corresponds. A specific destination can not be found in two sign lists at the same time. This rule results in the application of fixed vehicle routing strategy.

The sign lists concept has been used only at branch objects in the current SIMPLE++_AGVS library. By considering the possibility of bi-directional flow, it is desired to have the same approach at junctions, workstations, and parking stations.

Shortly, this feature must be provided for objects where a routing decision can be made. Therefore, the sign lists concept has also been applied to a number of other objects including `agvs_dock_station`, `agvs_station`,

agvs_junction2, and agvs_junction3. The sign lists are initialized based upon the shortest travel distances. Once they are filled, the sign lists are kept unchanged for the following simulations by turning the reset switch off. However, the user may alter some of these sign lists manually if the calculated shortest routes are not the routes used in the actual AGV system.

4.5.1.11. Bi-directional Flow Logic

Most AGV systems do not allow bi-directional flow due to the complexity of controlling the vehicle traffic. In some systems, a small section of the AGV path is dedicated for two-way traffic and the rest of the path is uni-directional. In the current SIMPLE++_AGVS library, the driving direction of a vehicle is not controlled. In other words, vehicles keep their initial driving direction (forward or backward) same during the entire simulation unless there is a terminal line on the AGV path. If there is a terminal line, the driving direction of a vehicle is reversed whenever the vehicle reaches the end of the terminal line.

A Boolean type attribute, named uni-directional, has been defined in all path elements in order to utilize the bi-directional flow logic. If this attribute is true, the path element does not allow reverse movement. Otherwise, the path element is bi-directional. Whenever a change of direction is considered on the path element, the attribute uni-directional is checked for the possibility of bi-directional move.

At an intersection, an AGV first searches the sign lists and finds which side it should move. If the destination is lying at the back side of the intersection, the possibility of bi-directional flow is checked. If bi-directional movement is possible, the driving direction of the vehicle is set as backwards.

4.5.1.12. Traffic Rules

In complex AGV systems, which has more than two vehicles and two-way traffic, vehicle traffic must be controlled at intersections. While an intersection is occupied by a vehicle, other vehicles are not allowed to enter the intersection. If there are two vehicles waiting at two different sides of an intersection, as the intersection zone becomes clear, one of the vehicles will occupy the zone earlier than the other vehicle. Traffic rules may be used to determine priorities at intersections.

A method, called `set_preference`, has been developed for controlling the vehicle traffic according to the specified rule. The user can select a traffic rule at a particular intersection by using the dialog window of the intersection object. The default traffic rule is the first-come-first-serve rule. The other rules that have been provided in the simulation package include vehicle-load-priority rule, left-side-first rule, right-side-first rule, up-side-first rule, and down-side-first rule.

The vehicle-load-priority rule selects the first vehicle according to the status of the vehicle and its load. For instance, a loaded AGV will have priority over an idle AGV. If an AGV is carrying a high priority order, it will have priority over the other AGVs which are carrying ordinary orders. The left-side-first, right-side-first, up-side-first and down-side-first rules sort the vehicles according to the approaching direction of the vehicle.

4.5.1.13. Dispatching Rules

The current `SIMPLE++_AGVS` library has provided several dispatching rule options. These rules were listed in Section 3.9. No more dispatching rules have been defined in this research study. In some AGV applications, dispatching rules are not employed. For example, vehicles may be assigned to a certain workstation so that they are directly returned to that workstation after completing a job without obeying any dispatching rule. In this case, none of the

dispatching rule options can be used. Therefore, a new menu option has been added to the dispatching rules menu in order to deactivate these rules.

4.5.2. Changes Related to Production System

Objects that are related to the Production system are ProdMgr, station, and Part. Many new features have been added to these production system-related objects. This section will briefly explain these new features and their functions.

4.5.2.1. Workstation States

A workstation can be in several states such as idle, operating, being setup, broken, or blocked. These states have been defined in the station object. Every time a station changes its status, the table showing the status of stations is updated. This table is monitored by the ProdMgr object in order to obtain the information regarding the current status of a workstation. This information is also used for collection of machine statistics.

4.5.2.2. Buffer States

Workstations interact with the AGV path through input and output buffers. An input buffer is represented by the entry place object in a station object (see SIMPLE++ User's Manual for the concept of placing objects in an object). An output buffer is represented by the exit place object in a station object. The input and output buffers may be in several states such as empty, occupied and not blocked, or blocked. These states have been defined for the purpose of collection of buffer statistics. Two tables have been placed into the station object

for keeping the status information of the input and output buffers, respectively. The methods for updating these tables have also been built in the station object.

4.5.2.3. Job Request Types

Three possible job request types have been supported by the proposed simulation package (see Section 3.3.) In the first type, the job types, their arrival times and quantities must be given at the beginning of the simulation. In the second and third types, jobs arrive to the system at specified time intervals. The difference between the second and third types is that in the third type, the operations plan for each job type must be provided by the user. After arrival of a job, all operations of the job are read from the operations plan, sequentially. In the second type, a job is a one transport request from an origin station to a destination station. The generation of the job requests can be arranged to be either from the origin object or from the destination object. The time between job arrivals can be defined as a constant or a random distribution value. A Generator object and a corresponding method have been built in order to activate the generation of jobs at specified time intervals and at specified workstations.

An AGV system may cover all these job request types in its structure. For example, one workstation may be requesting parts according to an operations plan, while, another workstation may be requesting parts from certain origin points (origin to destination). Therefore, a parameter called `jobRequestType` has been defined as a custom attribute of the station object. This attribute determines the job request type used for the specific station. It is not a global variable, but a station-dependent variable.

4.5.2.4. Independent Workstations

The independent workstation concept was introduced in Section 3.4. The difference between an ordinary workstation and an independent workstation is

that an independent workstation directly sends a finished job to its next destination without calling an AGV. In other words, the job is not sent to the corresponding agvs_dock_station object after which it is loaded to an AGV. The method used for transferring the jobs from the station object to the agvs_dock_station object has been modified in order to utilize the concept of independent workstations.

4.5.2.5. Machining Times

In the SIMPLE++_AGVS library, part specific machining times can be defined only in constant form. The proposed AGV simulator accepts both constant and random distribution values for the part specific machining times. The distribution type and the required parameters (e.g. mean, variance, etc.) must be specified by the user if a random distribution value is selected.

4.5.2.6. Job Priority

A Boolean type attribute, called highPriority, has been defined in the Part object for assigning priority to a specific job. Before the initialization of a simulation model, the user inputs the job types and their operations plans in the ProdMgr. In the proposed simulator, priority orders may also be determined at this stage. To make a job type a priority order, the user simply sets the attribute highPriority as true for that particular job type. If a job is from a high priority job type, it enters the system as a high priority order.

Job priority may also be assigned while the job is in progress. After certain operations, a job may become a rush order. Therefore, assigning priority is not only limited to the initialization stage, but also after each operation, jobs

can be checked if they get priority after the particular operation. To accomplish this, the format of the operations plan table in the ProdMgr has been modified so that the user can set priority for a specific operation of a particular job type.

4.5.2.7. Job States

A job may be in several states such as waiting to be processed, being setup, being processed, waiting for transport, and being transported. These job states have been defined in the proposed AGV simulator for the purpose of collection of job statistics. The states waiting to be processed, being setup and being processed represent the times spent in a workstation. The status waiting for transport represents the time spent in a dock station while the job is waiting for a vehicle. The status being transported represents the time spent for transporting the job from a station to another. The summation of all of these states is equal to the total time a job spends in the system.

At the beginning of the simulation, the initial status of a job may differ with respect to the job arrival type selected by the user. If the job starts at the input buffer of a workstation, the initial status of the job is set as waiting to be processed. On the other hand, if it starts at a dedicated arrival point in which the job does not need processing before entering the system (e.g. input storage), the initial status of the job is set as waiting for transport.

4.6. New AGV Application Objects

In addition to the modifications to the existing application objects, a number of new application objects have been developed during this research study. Figure 4.2. shows the appearance of the modified AGV library with the new application objects. The shapes of the existing application objects have

also been modified, because it is more convenient to use smaller icons for transferring the CAD drawing data into the simulation package.

The new application objects that have been built in this research include AGV Model Frame, Statistics, Route Manager, Maintenance Station, Battery Charge Station, and Model Builder. In the following sections, these new application objects will be introduced and their functions will be explained.

4.6.1. AGV Model Frame (AgvModel)

The first step for building a simulation model in SIMPLE++ is the creation of a copy of the Frame object and the placement of application objects into the new Frame object. A frame object called AgvModel has been created to be used as a base for building new simulation models. AgvModel contains the main AGV application objects (EventController, agvs-manager, ProdMgr, RouteMgr, and Stats) and the other AGV application objects such as station objects and path elements.



Figure 4.2. The modified AGV objects library

4.6.2. Statistics (Stats)

Stats is an application object that has been developed for collecting a variety of statistics. Figure 4.3. shows the structure of the Stats object. All

performance measures listed in Section 3.2 have been supported in the proposed AGV simulator via the Stats object. All these performance measures have been developed in this study and were not provided in the existing SIMPLE++ AGV library.

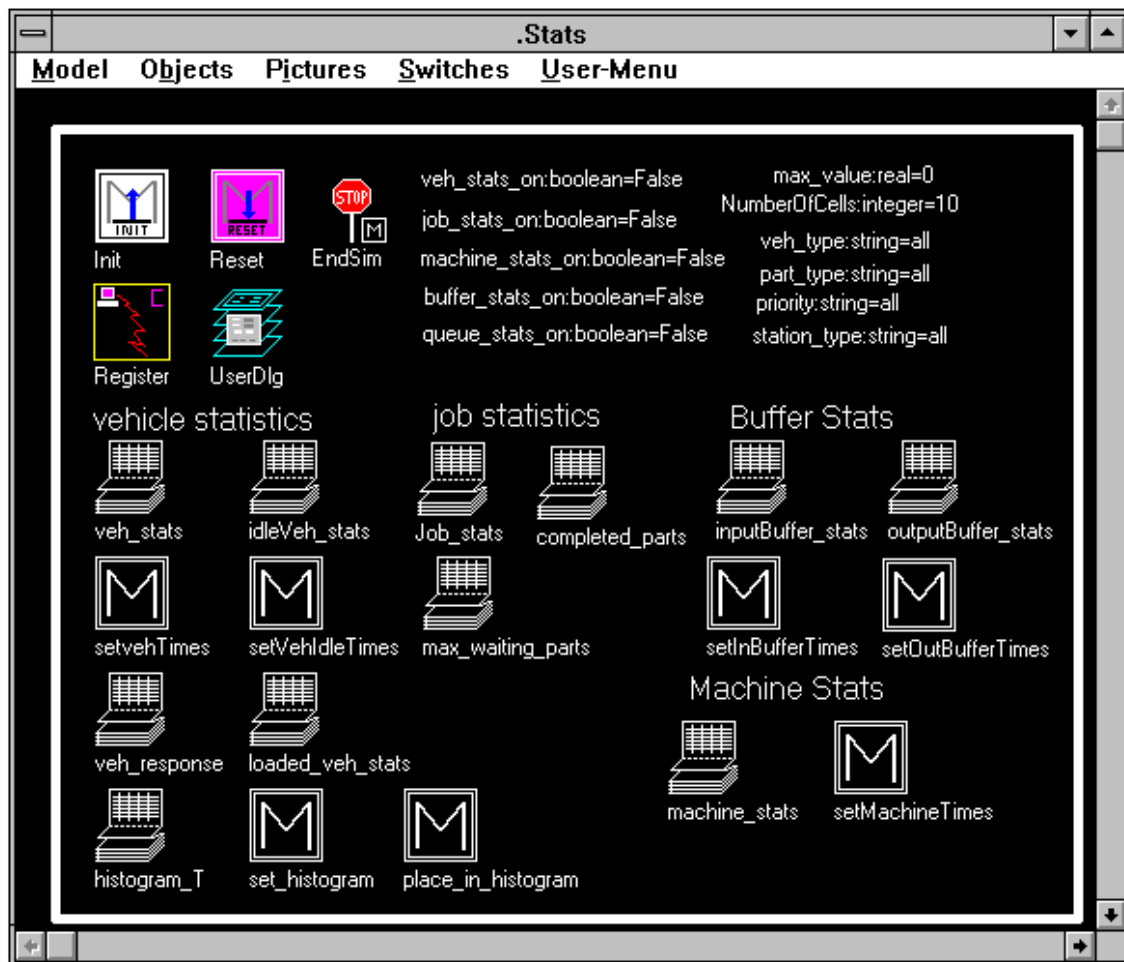


Figure 4.3. The structure of the Stats object

The statistics are classified as vehicle statistics, job statistics, workstation statistics, zone statistics, and buffer statistics. Each statistics option except zone statistics can be turned off by using the dialog window of the Stats object. The only statistics which has not been covered in the Stats object is the zone statistics. The zone statistics are collected at each zone object, separately.

Vehicle statistics are collected in a number of tables in the Stats object. These tables are `veh_stats`, `idleVeh_stats`, `veh_response`, `loaded_veh_stats`, and `histogram_T`. The `veh_stats` table shows the percentage of time spent in every vehicle status (idle, assigned, etc.) by the vehicle type. The `idleVeh_stats` table shows the percentage of time spent in every idle vehicle status (moving, parking, atMachine) by the vehicle type. The `veh_response` table shows the total vehicle response time for each transport request. The `loaded_veh_stats` table shows the frequency of the number of loads carried by vehicles.

The last two tables (`veh_response` and `loaded_veh_stats`) have been made optional, because recording all these statistics takes too much computer memory. Instead, the `histogram_T` table may be used for collecting data in the form of a histogram. To obtain histogram data, the user must specify the maximum value and the number of cells in the histogram along with the vehicle type to be reported. These values are set in the dialog window of the Stats object.

The `job_stats` table in the Stats object contains the records of times spent in each job status (moving, processing, etc.) for each job entered the system. Similar to the vehicle statistics, the collection of all job statistics may not be necessary, and therefore, this table has also been provided as optional. The `histogram_T` table can be used for obtaining histograms showing the distribution of job states by time and by job type. Again, the maximum value, the number of cells in the histogram, and the job type to be reported must be specified by the user.

The total number of jobs completed from each job type and the maximum number of jobs waiting for an idle vehicle are other job statistics that have been provided in the Stats object. The maximum number of jobs waiting for an idle vehicle can be reported either for a specific station or for the overall system.

Workstation statistics are collected in the machine_stats table in the Stats object. The machine_stats table shows the percentage of time spent in each machine status (idle, operating, setup, broken, and blocked) for each workstation in the system. The initial status of a machine is idle. If a workstation does not have a setup operation, the total setup time percentage reported in the machine_stats table is zero. Similarly, if breakdowns are not activated, the total broken time percentage will also be zero.

Input buffer and output buffer statistics are stored in the inputBuffer_stats and outputBuffer_stats tables, respectively. These tables show the percentage of time spent in each buffer status (empty, occupied and not blocked, and blocked) for the buffers of a specific workstation.

The zone statistics are not included in the Stats object, but rather in every zone object itself. The zone statistics provided in a zone object includes maximum length of time spent in a zone, histogram for waiting times in queue, total number of vehicle visits to a zone, and the number of vehicle visits in which a blockage occurred.

The maximum length of time in a zone is stored in the max_block table in every zone object (e.g. agvs_straight). The total number of vehicle visits and the number of visits with blockage are shown in the zone_stats table in all zone objects. These two statistics can be reached by pressing the Zone Statistics button in the dialog window of a zone object. The other zone statistics regarding vehicle blockage at zones have been provided in the RouteMgr object which will be explained in the following section.

4.6.3. Route Manager (RouteMgr)

RouteMgr is an application object that is used for controlling the vehicle traffic and monitoring the vehicle blockage in an AGV system. When a blockage occurs, the RouteMgr object is notified. The blocked vehicles and the blocked

zones are stored into tables so that dynamic routing actions can be programmed by the user. Additionally, some of the zone statistics which are related to the vehicle blockage are collected in the RouteMgr. Figure 4.4 shows the structure of the RouteMgr object.

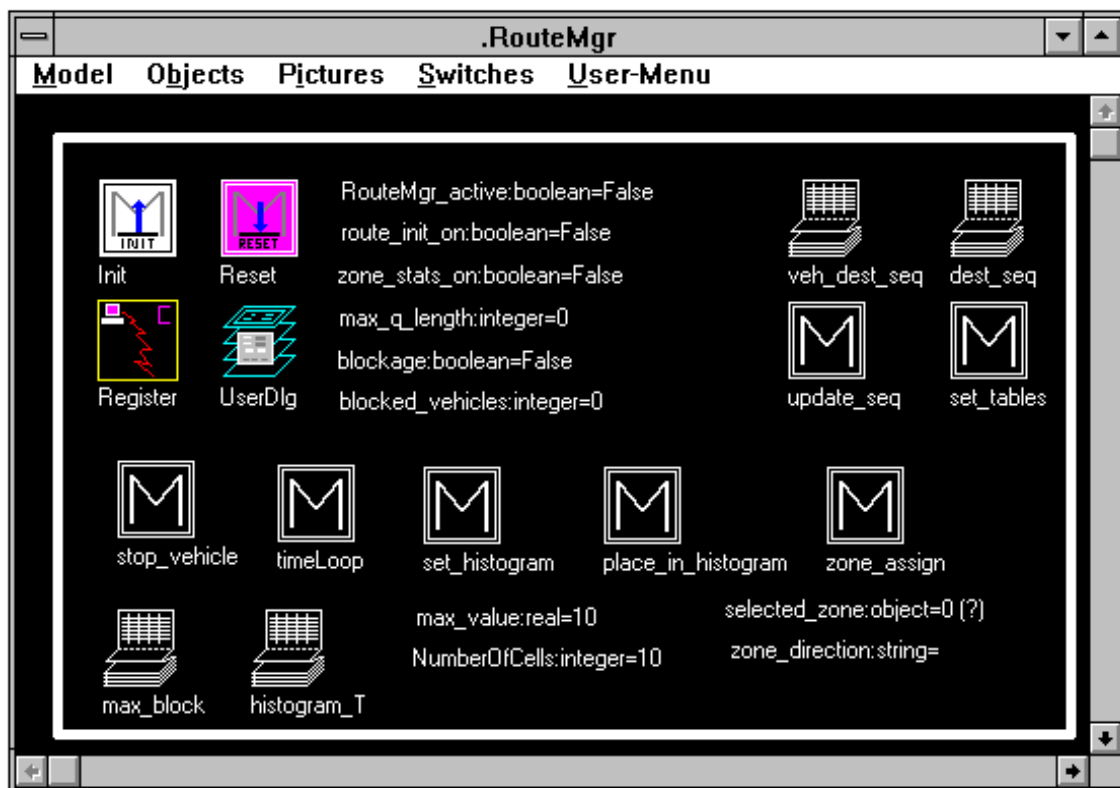


Figure 4.4. The structure of the RouteMgr object

When an AGV is blocked, the stop_vehicle method in the RouteMgr is performed. This method sets the vehicle status as blocked and updates the tables for vehicle statistics in the Stats object. It also records the names of the blocked vehicle and the blocked zone. The veh_dest_seq table in the RouteMgr object shows the current destination sequence list of all vehicles in an AGV

system. Destination sequence list of a vehicle is a list of zone objects that will be visited by the vehicle until reaching to its current destination. The `veh_dest_seq` table is updated by the `update_seq` method (see Figure 4.4.) every time a vehicle enters a new zone object.

The `veh_dest_seq` table has been developed for the purpose of programming dynamic vehicle routing strategies. Routing decisions can be made based upon the blockage information provided in the `RouteMgr` object. By reaching a decision point, an AGV requires the list of currently blocked zones from the `RouteMgr`. If there are any blocked zones, the AGV, then, requires its current destination sequence list in order to see if there is a blocked zone on its way to the current destination. If the current path is blocked, a method for changing the current path may be performed (if dynamic routing is possible). There is not such a method defined in the `RouteMgr` object. However, it can be placed into the `RouteMgr` object if a specific dynamic routing strategy is going to be tested by the simulator.

Zone statistics provided in the `RouteMgr` object includes histogram of waiting times in queue at a particular zone, the maximum blocking time and the maximum queue length in the overall system. For the histograms, the number of cells and the maximum value in the histogram, and the zone name must be specified by the user. The `max_block` table in the `RouteMgr` object shows the maximum blocking time occurred in the overall system by specific zone name and vehicle type. An attribute named `max_q_length` has been defined for storing the maximum queue length observed in the system. The phrase queue length represents the number of vehicles waiting behind a blocked vehicle. The value of the attribute `max_q_length` is updated whenever a larger queue length is observed. Zone statistics can be turned off to save computer time.

4.6.4. Maintenance Station (Maintenance)

Maintenance is an application object which represents a maintenance station in a physical AGV system. It is a path element that has to be connected to the AGV path. A broken AGV is sent to the Maintenance object if the breakdown is a type A breakdown (see section 4.5.1.4 for breakdown types). The AGV waits at the Maintenance for the time specified in the dialog window of the AGV object. By the end of this time, the AGV is sent back to the main path. The structure of the Maintenance object is shown in Figure 4.5.

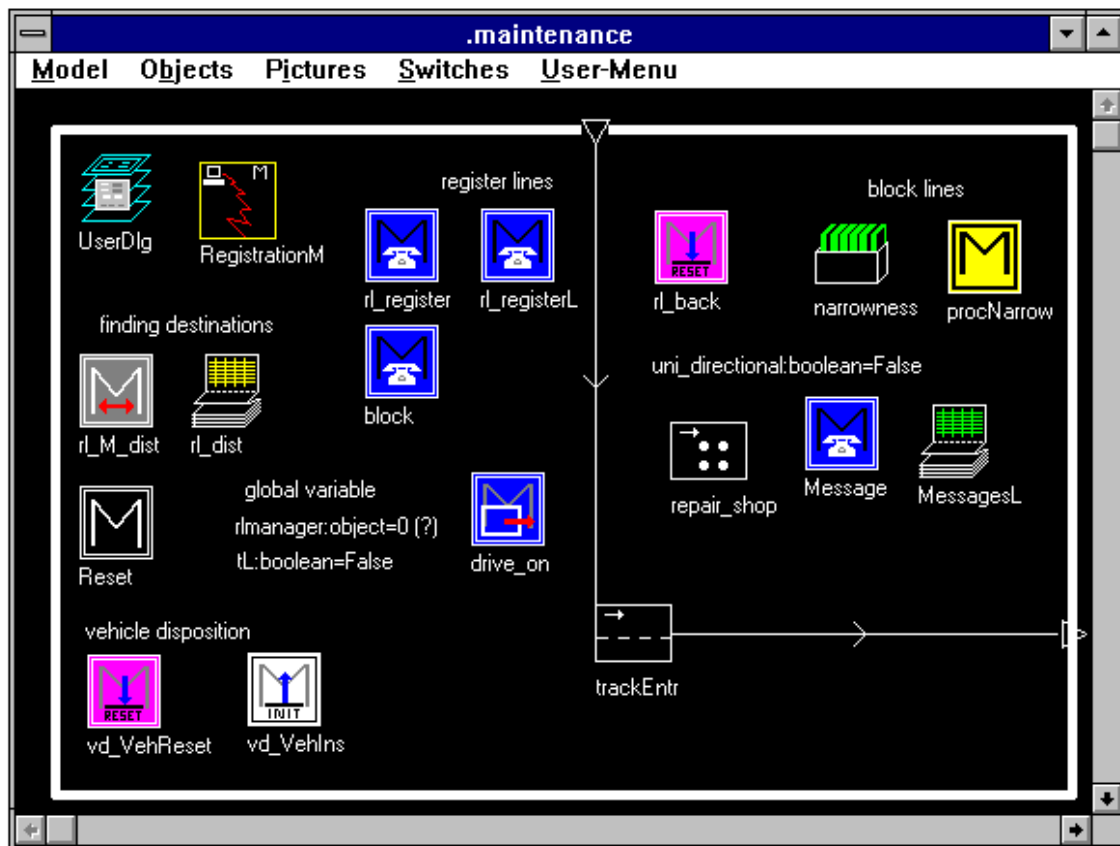


Figure 4.5. The structure of the Maintenance object

The maximum number of vehicles that can be sent to the Maintenance object at any time must be specified by the user. The default value is unlimited. The Maintenance object must be placed into the AgvModel frame object so that vehicle breakdowns can be simulated.

4.6.5. Battery Charge Station (bat_charge_station)

The bat_charge_station is an application object which represents a battery charge station in a physical AGV system. It is a path element that has to be connected to the AGV path. The structure of the bat_charge_station object is shown in Figure 4.6.

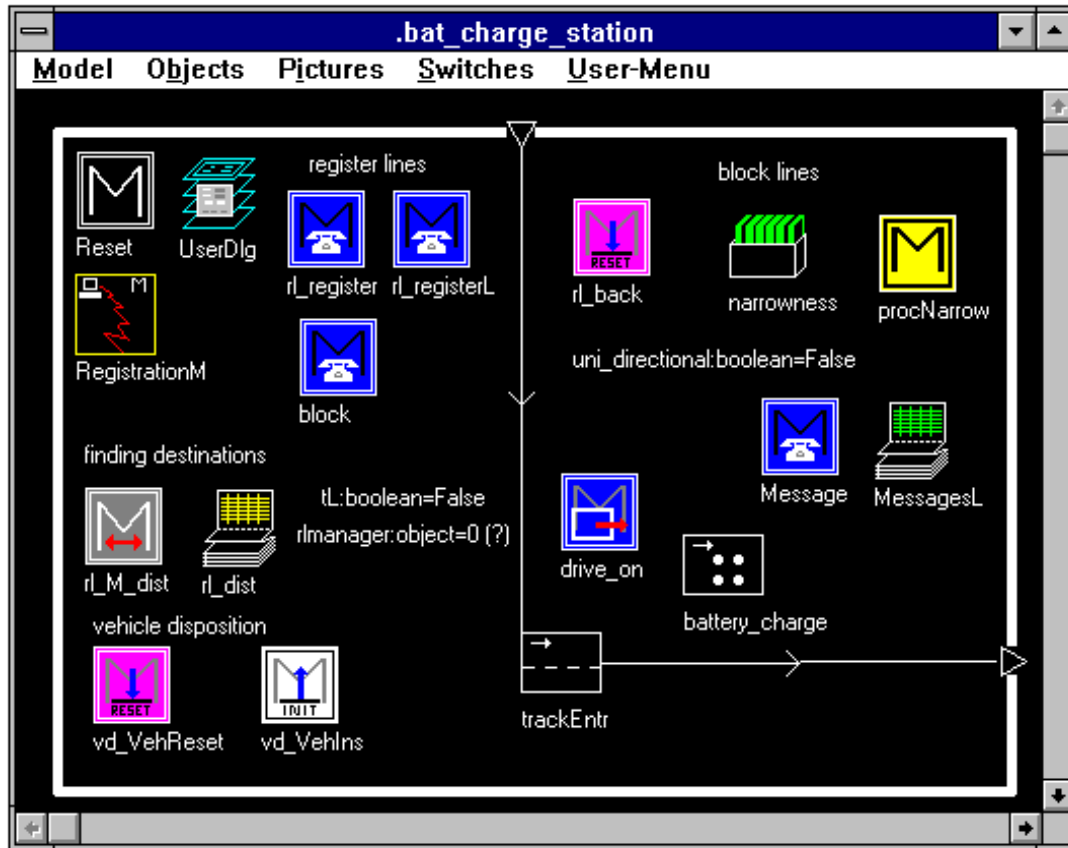


Figure 4.6. The structure of the bat_charge_station object

When the battery charge of an AGV reaches the specified safety level, the AGV is sent to the bat_charge_station object. If the normal battery charging technique is utilized, the charging time(duration) is calculated by using the following formula:

$$duration = \frac{\text{maximumBatteryCapacity} - \text{currentChargeLevel}}{\text{chargingCurrent}}$$

Here, the difference between the maximum battery capacity (in Amperes) and the current charge level of the AGV (in Amperes) gives the total charge to be refilled (in Amperes). The result is divided by the charging current (in Amperes/hour) to obtain the time spent in the battery charge station. If the battery exchange operation is utilized, the changing time of the battery is input by the user.

The maximum number of vehicles that can be located at the bat_charge_station must be specified. The default value is unlimited. The bat_charge_station object must be placed into the AgvModel frame object in order to simulate the battery charging feature of AGV systems. However, even if it is not placed, the opportunity charging feature can be utilized as the battery charging technique in the simulation model.

4.6.6. Model Builder (Build)

Build is an application object which is used for building a new AGV model. The structure of the Build object is shown in Figure 4.7. In SIMPLE++, a simulation model can be developed by placing several application objects into an empty frame. The application objects must be connected in a sequential manner.

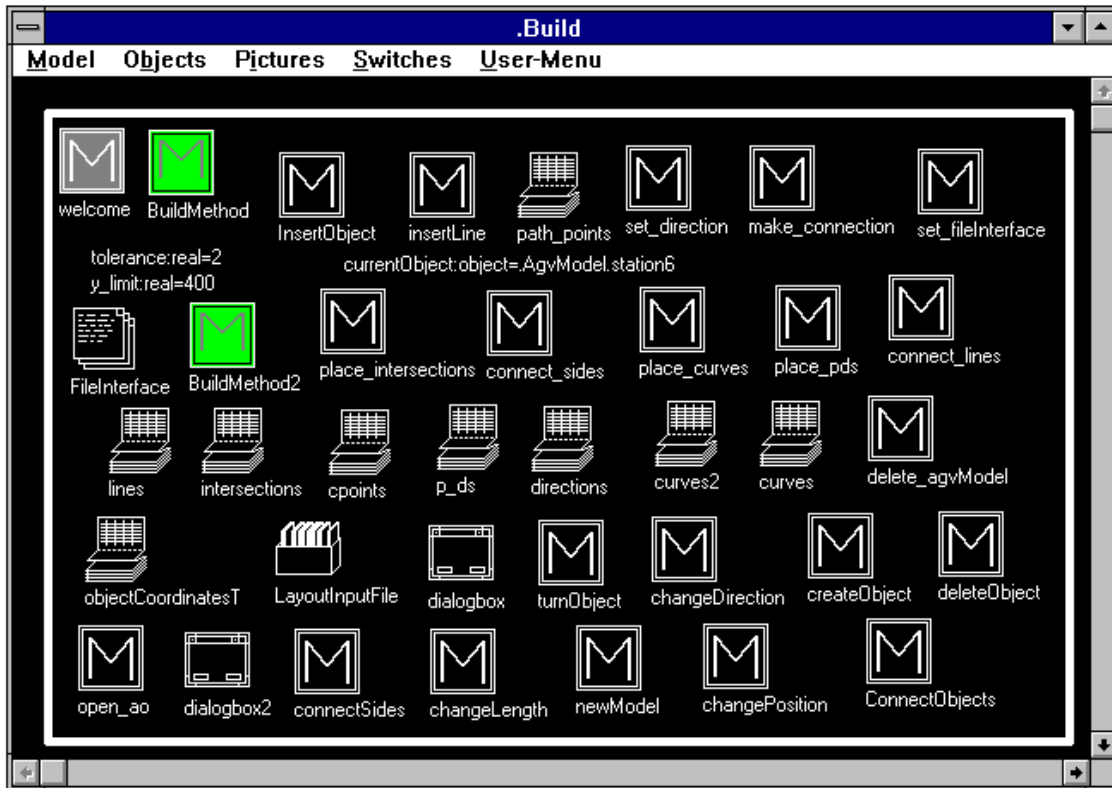


Figure 4.7. The structure of the Build object

Two ways of building a new AGV model have been offered in the proposed AGV simulator: manual and automatic. In the manual method, the AGV application objects are copied from the AGV library into an empty frame, and then connected to each other, sequentially. In the automatic method, the AGV layout, the coordinates of the control points, and the directions of the path sections are read from a CAD drawing data output.

The automatic model building method is implemented by the use of Build object. The method named BuildMethod in the Build object has been developed for translating a CAD drawing of an AGV system into a corresponding AGV model in SIMPLE++. The FileInterface object in the Build object is an interface between SIMPLE++ and a standard CAD output file (.dxf file).

The user specifies the name of the CAD file in the dialog window of the FileInterface object, and the BuildMethod reads this file line by line in order to create the new model in the AgvModel frame object. Transferring CAD drawings into SIMPLE++ will be explained in detail in the Appendices.

4.7. Conclusion

In this section, the existing features of the SIMPLE++_AGVS library have been introduced and the enhancements made in this research study have been reported. These enhancements include adding new methods, new pictures, new tables to the existing application objects, and building new application objects (e.g. Stats object).

Chapter V

5. ILLUSTRATION OF PROCEDURES FOR USING AGV SIMULATOR

5.1. Introduction

In this section, the procedures for building a new simulation model, initializing the model, and running the simulation will be illustrated. An example AGV model has been developed in order to demonstrate these procedures.

5.2. An Example AGV Model

A sample AGV model has been constructed for the purpose of illustrating the initialization and execution of a simulation model. The model is taken from Pritsker's [Pritsker, 1995] SLAM model that is developed as an example of simulating AGV systems.

5.3. Description of the Model

The model consists of six machines, one fixturing station, a fixturing space, a parking station, and two vehicles which are identical and have 4 ft./second loaded speed and 4.5 ft./second empty speed. The layout of the sample AGV model is given in Figure 5.1. Vehicle flow is uni-directional along all the segments, except the spur segment in front of the fixturing station. Vehicles are allowed to travel in both directions on this spur. Hence, only one vehicle can enter the spur in order to prevent traffic congestion.

(Figure 5.1. is going to be placed here)

Neither the vehicles nor the machines are subject to breakdowns. Vehicles do not change or charge batteries. Jobs are carried on a fixture. The fixture is not taken off until the machining of the job is finished. Each machine, and the fixturing station can process only one job at a time. There is not a setup time associated with the machining time. There is only one job type and one processing type in the system. All six machines do the same process on the same type of part.

Jobs arrive to the system exponentially with a mean of 436 seconds. Arriving jobs need to be fixtured at the fixturing station. Loading the job on a fixture takes 220 seconds. If the fixturing station is occupied, the job waits in a queue (file 1). After fixturing, the job waits in the fixturing space from where it is transferred to the AGV path. The holding capacity of the fixturing space is five units. If the fixturing space is full, the fixturing station does not accept any more jobs until one more space is available at the fixturing space.

A part can be sent to the AGV path only if a machine is available; otherwise, the part waits in the fixturing space. The machines are numbered from one to six. If machine one is available, it is allocated first. The number of the machine determines its priority regarding the job allocation.

Upon allocation of a part to a machine, a free vehicle is assigned to the job. If there is not an idle vehicle, the part waits in a queue which is based on the first-come-first-serve rule. The assigned vehicle picks up the part and delivers it to the allocated machine. Loading a part onto a vehicle or unloading the part from the vehicle takes 45 seconds.

Machining time for a part is machine specific and is triangularly distributed with a minimum value of 960, a modal value of 1860, and a maximum value of 2720 seconds. Upon completion of transporting a part to a machine, if no other parts are waiting, the AGV is sent to the parking station.

When the processing of a part is finished, the part is carried back to the fixturing station. If the fixturing station is occupied, the part is placed into the same queue where the first arriving parts are placed (file 1). In this queue, the processed parts have the priority against the first arriving parts. Unloading a part from the fixture takes 180 seconds.

The layout consists of seven pick up and deposit points, one parking station, and five intersection points. The length and the flow type (uni/bi-directional) of the sections between these locations are given in Table 5.1.

Table 5.1. Length between location points in the layout

FROM	TO	LENGTH (ft)	FLOW TYPE
fixturing	Intersection-1	20	bi
Intersection-1	Intersection-2	82	uni
Intersection-2	Parking	27	uni
Parking	Intersection-3	27	uni
Intersection-2	Intersection-3	16	uni
Intersection-3	Intersection-4	13	uni
Intersection-4	Machine-1	35	uni
Machine-1	Machine-2	35	uni
Machine-2	Machine-3	35	uni
Machine-3	Intersection-5	35	uni
Intersection-4	Machine-4	75	uni
Machine-4	Machine-5	35	uni
Machine-5	Machine-6	35	uni
Machine-6	Intersection-5	75	uni
Intersection-5	Intersection-1	115	uni

5.4. Building the Simulation Model

The first step in building a model is drawing the AGV layout in a CAD environment. The CAD drawing contains the layout data including distances between destination points, pick up and deposit point coordinates, directions, and control point coordinates. Details and procedures for translating an ordinary CAD drawing into a SIMPLE++ compatible drawing will be explained in the Appendices. The output file (.dxf file) of a CAD drawing is transferred to SIMPLE++, and by selecting the BuildModel method, the corresponding AGV model is created automatically in the SIMPLE++ AGV simulator. The appearance of the example model after this translation can be seen in Figure 5.2.

(Figure 5.2. is going to be placed here)

This is not a complete model that can be run by the simulation program. The following steps must be executed in order to complete the model:

- 1) Place the workstation objects and connect them to the corresponding dock stations.
- 2) Set the machining times and workstation specifications.
- 3) Set the AGV system related specifications.
- 4) Define the part types, operations plans, and priorities.

- 5) Set the simulation end time and initialize the model.
- 6) Set the job arrival times.
- 7) Set vehicle specifications.
- 8) Turn the AGV reset switch off and run the simulation.

These steps are applied for running the simulation without obtaining the simulation results. The procedures for obtaining the statistics outputs will be explained in the next chapter. In the following sections, the above procedures for building an AGV model will be demonstrated.

5.4.1 Placing and Connecting the Workstation Objects

Six machines, a fixturing station, a fixturing space, an enter station, and an exit station have been copied into the example model from the objects library and have been connected to the corresponding dock stations manually. The new appearance of the example model is shown in Figure 5.3. The picture of a workstation object can be changed in order to represent different types of workstations. The coordinates of a workstation object is not important for the simulation. On the other hand, the coordinates of the dock stations are important and are used for calculating the distances between destination points. Some workstations are connected to the path through dock stations while other workstations are independent from the path. In this model, the fixturing station, the fixturing space, and enter and exit stations are independent workstations.

(Figure 5.3. is going to be placed here)

5.4.2. Setting the Machining Times and Workstation Specifications

Each workstation serves a special purpose (e.g. machining, fixturing, etc.). The machining times, operations list, and capacities must be specified for each workstation. The user can open the dialog window of a station object by a double click on that object. The dialog window of a station object is shown in Figure 5.4. At the “station type” prompt, the station type of the workstation (e.g. fixturing) can be specified. At the “station no” prompt, the station number can be specified. The total capacity of a workstation must be specified if the station has

a limit for holding a certain number of parts at a time. The total capacity for the fixturing station and the six machines is one unit and the total capacity for the fixturing space is five units.

(Figure 5.4. is going to be placed here)

The machining times in the example model are all machine specific and can be specified at the “work place definition” prompt. For the fixturing station, the machining time is determined according to the job status. If the job has just arrived, the loading time is 220 seconds. If the job is leaving the system, the unloading time is 180 seconds.

5.4.3. Setting the AGV System Related Specifications

The number of vehicles, the vehicle dispatching rule, and general specifications regarding the vehicle control system can be set in the dialog window of the agvs_manager object. The dialog window of the agvs_manager object is shown in Figure 5.5. Empty and loaded vehicle speeds are set in the dialog window of the AGV object. The dialog window of the AGV object is shown in Figure 5.6. The maximum number of vehicles allowed in the parking station is specified in the dialog window of the agvs_station (parking station) object. The dialog window of the agvs_station object is shown in Figure 5.7.

(Figure 5.5. is going to be placed here)

Show

Name: ☐ Nr.: 00003 ☒ Disrupted ☒ Pause

Charge [Ah]: ☐

Capacity [Ah]: ☐ Reserve [Ah]: ☐

Char. Current [A]: ☐ **Charge ctrl** ☐

Basic cons. [A]: ☐ Driving cons. [A]: ☐

Cust. attributes:

shiftWork=False (boolean) C ni
 loadTime=45.0000 (const) (time) C ni
 break_on=False (boolean) C ni
 break_interval=1:39.5988 (1,1:40.0000,1.0000,0.0000,0.0000) [normal] (time) C ni
 repairTime=1:38.8850 (1,1:40.0000,1.0000,0.0000,0.0000) [normal] (time) C ni
 fd_idleStatus= (string) C ni
 IdleMovingTime=0.0000 (const) (time) C ni
 transport_no=0 (integer) C ni
 vehicle_type= (string) C ni
 repairStayTime=10.0000 (const) (time) C ni
 origin_station=? (object) C ni
 unloadTime=45.0000 (const) (time) C ni
 loaded_speed=4 (speed) C ni
 normal_speed=4.5 (speed) C ni
 charge_control=False (boolean) C ni
 charge_con_interval=2:00:00.0000 (const) (time) C ni

Figure 5.6. The dialog window of the AGV object.

5.4.4. Defining the Part Types, Their Operations Plans and Priorities

Part type information is input from the tables in the ProdMgr object. The dialog window of the ProdMgr object is shown in Figure 5.8. By pressing the button named Master in the dialog window, a table appears. In this table, part types, their operation plans, priorities, and the vehicles that are assigned to each part type can be specified. The appearance of this table is shown in Figure 5.9.

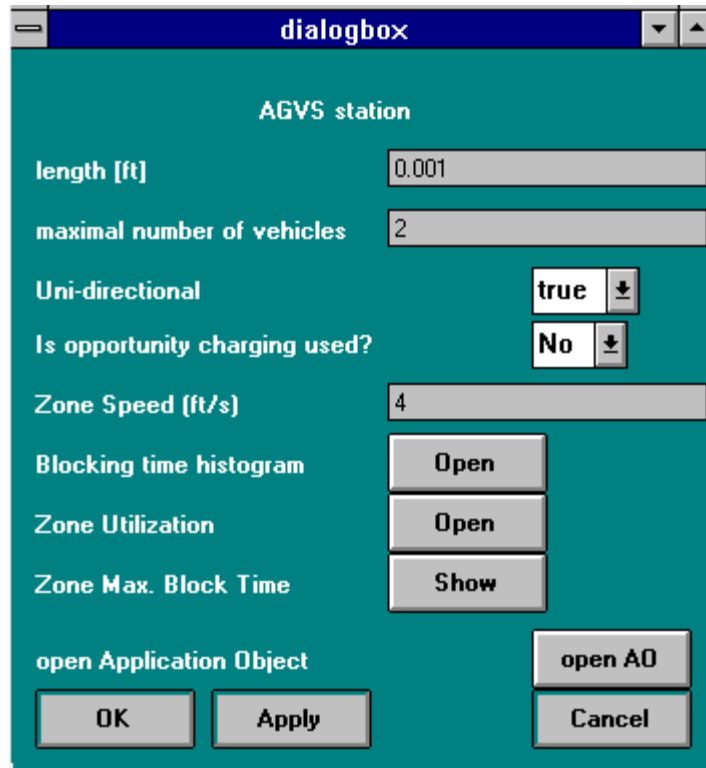


Figure 5.7. The dialog window of the agvs_station object



Figure 5.8. The dialog window of the ProdMgr object

In the example model, there is only one part type. Its operations plan is specified as: entersystem-load to the fixture-wait-process-unload from the fixture-exitsystem. According to the rules specified for the example model, the processed parts have priority against the first arrived parts at the fixturing station. Therefore, the process of unloading the part from the fixture has a priority sign in the operations plan table. As default, all vehicles are assigned to all part types and to all workstations. However, the user may select only certain vehicles to serve for specific part types or workstations.

	string 0 Entity type	table 1 OperationsPlan_L	list 2 vehicles	boolean 3 highPriority
0	part1	plan1	list21	
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				

Figure 5.9. The table for setting the part type information

5.4.5. Setting the Simulation End Time and Initializing

The example model has been run for 89400 seconds (approximately 1 day). The ending time of a simulation can be set in the dialog window of the EventController object. The dialog window of the EventController object is shown in Figure 5.10. By pressing the Init button in the EventController, all Init methods in all objects are called, distances between destination points and the sign lists are registered, vehicles are placed into the parking station, and all workstations, job type definitions and their operations plans are registered.

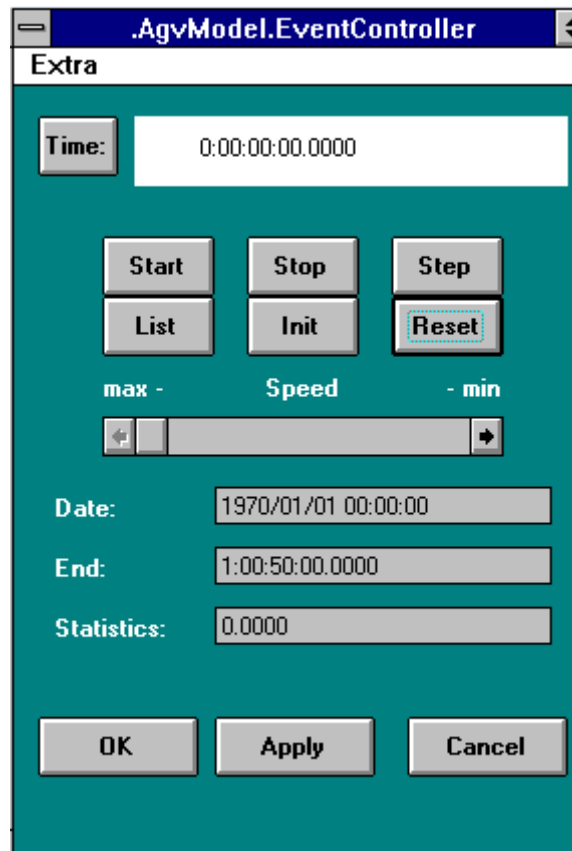


Figure 5.10. The dialog window of the EventController object

5.4.6. Job Arrival Times

In the example model, jobs come to the system exponentially and from the enter station. In the dialog window of a station object, a pop up menu has been provided for setting the part arrival times for a specific part type. There is only one part type in the sample model. The user selects this part type from the pop up menu and another dialog window appears. In this dialog window (see Figure 5.11.), the job arrival times for the selected part type can be specified. For the example model, the user must select the exponential distribution and type 436 seconds at the “Interval” prompt.

The screenshot shows a dialog window titled ".AgvModel.station8.Generators.part1". It has a teal background. At the top, there is a "Name" field with "part1" and a "Locat." button. Below this, there are four rows of settings, each with a label, a dropdown menu, a text field, and a checkbox:

Label	Dropdown	Text Field	Checkbox
Start:	const	0.0000	☒
Stop:	const	0.0000	☒
Interval:	negexp	1, 7:16.0000, 0.0000, 0.0000	☐
Duration:	const	0.0000	☒

At the bottom, there are three buttons: "OK", "Apply", and "Cancel". A vertical scrollbar is on the right side of the dialog.

Figure 5.11. The dialog window for setting the part arrival times

5.4.7. Setting the Vehicle Specifications

After the initialization, the vehicle types (tow, unit, etc) and other vehicle specifications such as battery constraints, vehicle breakdowns, etc. can be specified. For the example model, no such specifications have been needed. However, the user must make sure that the switches related to breakdowns and battery charging have been turned off in the dialog window of each AGV object.

5.4.8. Turning the AgvReset Switch off and Running the Simulation

If the user does not want to redo all the vehicle settings everytime a new simulation is started, the AgvReset menu option in the dialog window of the agvs_manager object must be set as False. Otherwise, the Reset button in the EventController object will delete all the vehicles at the end of the simulation.

Now, the simulation is ready to run. By pressing the Start button, the simulation can be started. The Stop button is used to pause the execution of the program. The Step button executes one step (one method) at a time., The dialog window of the Stats object must be used in order to obtain statistics outputs. The next chapter will describe how to use the statistics options in the Stats object.

5.5. Conclusion

This section has provided some introductory demonstrations for the purpose of familiarizing the user with the usage of the generic AGV simulator. In the next chapter, more details will be given regarding the output generation capabilities of the simulator.

Chapter VI

6. STATISTICS AND MODEL VALIDATION

6.1. Introduction

In this section, first, the statistics options provided in the simulation package will be demonstrated and the results obtained from the example model given in chapter 5 will be reported. The results may not be exactly the same results as those given in the Pritsker's SLAM model. This is because of the differences between the modeling approaches used in two different models. Second, the capabilities of the generic AGV simulator, which have been described in Chapters 3 and 4, will be demonstrated with some examples.

6.2. Validation of the Outputs of the Example Model

All statistics options except the zone statistics are provided in the Stats object. The user can turn each statistics option on or off in the dialog window of the Stats object. The appearance of the dialog window of the Stats object is shown in Figure 6.1.

6.2.1. Job Statistics

The measures related to the jobs can be gathered by turning on the job statistics in the Stats dialog window. There is an additional pop-up menu for the job statistics which includes a list of job statistics options.

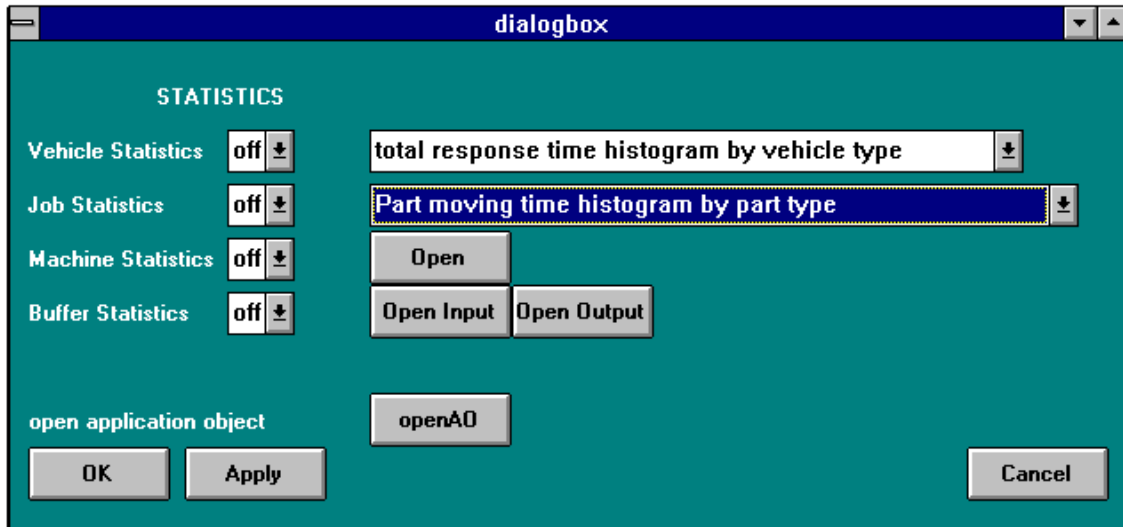


Figure 6.1. The dialog window of the Stats object

These job statistics include the number of completed parts; histograms of moving, processing, and setup times for a part type; the maximum number of waiting parts for a transportation; a histogram of the total time spent in the system. In the simulation of the example model, the observed number of completed parts is 190, which is very close to the result obtained with Pritsker's SLAM model (188 parts).

The total time spent in the system is output in the form of a histogram. The maximum value and the number of cells of the histogram can be defined by the user. Additionally, the average, and the minimum and maximum values observed during the collection of the histogram data are also provided.

The resultant histogram for the total time spent in the system is shown in Figure 6.2. The average time in the system is 3746 seconds (based on the mid points of the histogram values), and the standard deviation of the time in the system is 1019 seconds. Additionally, the observed maximum and minimum values are 6930 seconds and 2055 seconds, respectively.

.AgvModel.Stats.histogram_T			
File Edit Format Extra			
Formula			
	real 1	real 2	integer 3
string 0	interval_start	interval_end	frequency
61	3000.00	3049.99	3
62	3050.00	3099.99	3
63	3100.00	3149.99	3
64	3150.00	3199.99	2
65	3200.00	3249.99	1
66	3250.00	3299.99	6
67	3300.00	3349.99	5
68	3350.00	3399.99	3
69	3400.00	3449.99	4
70	3450.00	3499.99	3
71	3500.00	3549.99	2
72	3550.00	3599.99	3
73	3600.00	3649.99	1
74	3650.00	3699.99	1
75	3700.00	3749.99	7
76	3750.00	3799.99	3
77	3800.00	3849.99	3
78	3850.00	3899.99	2
79	3900.00	3949.99	4
80	3950.00	3999.99	6
81	4000.00	4049.99	3
82	4050.00	4099.99	2
	4100.00	4149.99	1

Figure 6.2. The histogram for the total time spent in the system

The processed jobs and first arriving jobs wait in a queue (file 1) in front of the fixturing station. The average waiting time in this queue has been reported as 481 seconds which is consistent with the model data since there are 400 seconds of fixturing time required (220 seconds to load part on the fixture and 180 seconds to unload the part from the fixture). The maximum length of this queue has been observed as 12 units.

Another output provided in the pop-up menu for job statistics is the maximum number of waiting parts at a particular station and by specific part type. The user can obtain the maximum number of loads waiting for a vehicle at a particular station type by specifying the name of the station type in the Stats dialog window. For example, the observed maximum number of loads waiting for a vehicle at the fixturing station and at the six machines are both two units. It is also possible to report the maximum number of waiting parts in the overall system. The name of the station type must be specified as “all” in order to obtain this report. The resulting maximum number of waiting parts is three units in the overall system which is same as the result obtained with the SLAM model.

6.2.2. Vehicle Statistics

The measures related to the vehicles are collected by turning on the vehicle statistics in the Stats dialog window. There is an additional pop-up menu for the vehicle statistics that include several vehicle statistics options such as time percentage in each vehicle status; time percentage in each idle vehicle status (moving, parking, atMachine); histograms of total response time, waiting response time, traveling response time; and the distribution of number of loads on a vehicle.

The histogram of the total vehicle response time can be collected for a specific station type. For example, the histogram showing the total vehicle response time for fixturing station is shown in Figure 6.3. The total vehicle response time is equal to the part waiting time for a vehicle (including both the waiting time before vehicle assignment and the empty vehicle traveling time). Based on the mid points in the histogram data, the average part waiting time at the fixturing station is 85 seconds, which is close to the result obtained by the SLAM model (81 seconds). Similarly, the observed average part waiting time at the six machines is 58 seconds.

The total response time is equal to the sum of the part waiting time before vehicle assignment and the empty vehicle traveling time to the workstation. It is also possible to report the part waiting time and the empty vehicle traveling time in two separate histograms. The average part waiting time before vehicle assignment was reported as 24 seconds in the SLAM model. It has been observed as 25 seconds in the generic AGV simulator.

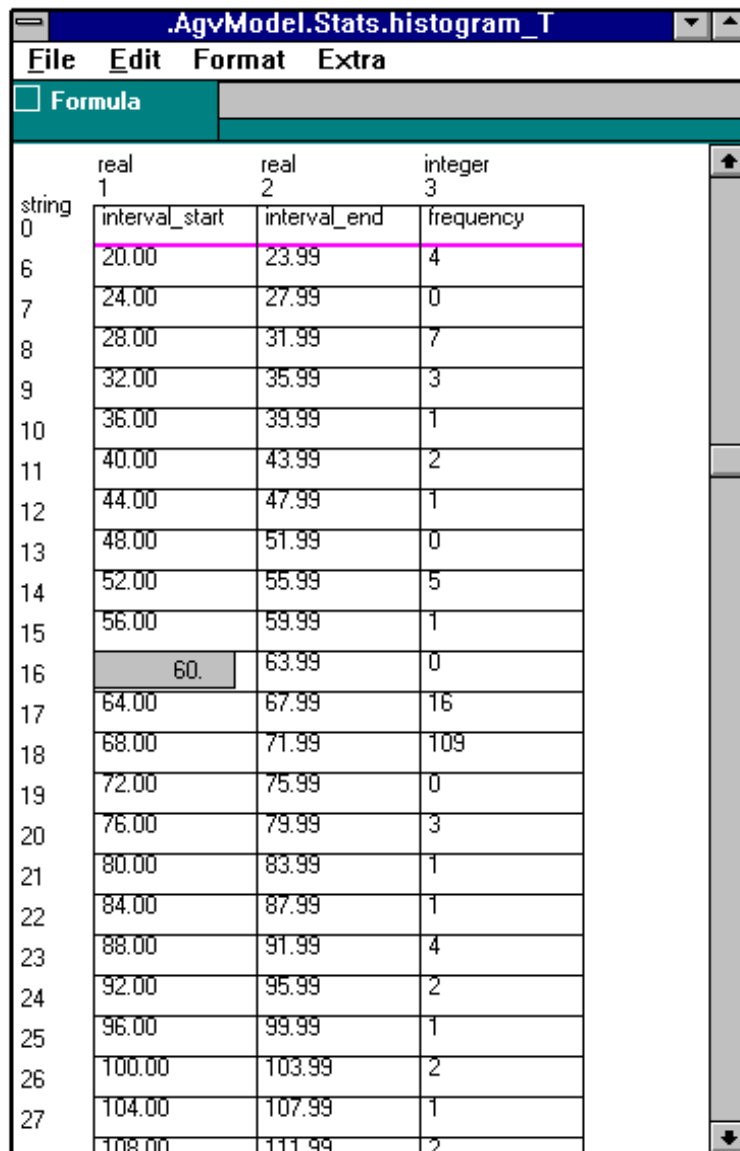


Figure 6.3. The histogram for the total vehicle response time at fixturing station

Vehicle states observed in the example model are, namely, idle, assigned, occupied, and blocked. The assigned status represents the empty traveling time to load a part. The occupied status represents the full traveling time to unload a part. The final vehicle utilization report is shown in Figure 6.4. The sum of the occupied and assigned states represents the total productive time of a vehicle. Hence, the total productive time of the first vehicle is 34.87%, and the total productive time of the second vehicle is 33.49%. Multiplying these numbers by the time period over which statistics were computed, 89400 seconds, yields a result that the first and second vehicles were productive 31174 and 29940 seconds, respectively.

.AgyModel.Stats.veh_stats								
File Edit Format Extra								
Formula								
	object 1	real 2	real 3	real 4	real 5	real 6	real 7	real 8
string 0	vehicle	idle_time	assigned	occupied	charging	failed	blocked	total
1	.agv:1	64.34	12.99	21.88	0.00	0.00	0.80	89400.00
2	.agv:2	65.76	11.66	21.83	0.00	0.00	0.75	89400.00
3								
4								
5								

Figure 6.4. The final vehicle utilization report

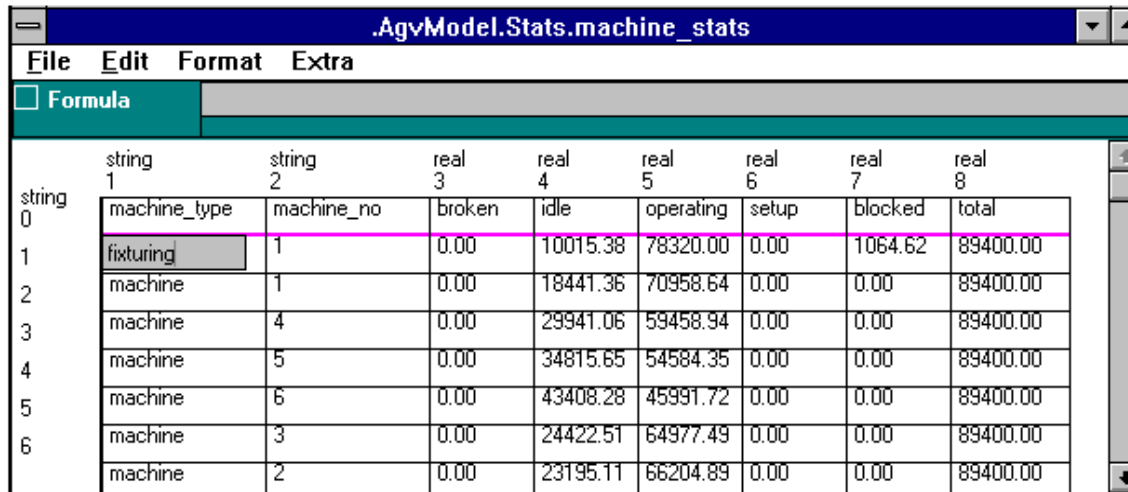
The blockage times of the two vehicles have been observed as 0.8% and 0.75% of the total simulation time. The total idle time is divided into two categories: parking and traveling. The final report for the idle vehicle states is shown in Table 6.1. According to the table, the total idle time of the first vehicle is 57522 seconds. 75% of this total idle time is at the parking station, while 25% of the total idle time is traveling idle. The results for the second vehicle are similar to the results for the first vehicle.

Table 6.1. Idle vehicle status report

vehicle	parking	moving	atMachine	total
.agv:1	75.04	24.96	0	57522.01
.agv:2	77.49	22.51	0	58786.59

6.2.3. Machine Statistics

The statistics related to the machines can be collected by turning on the machine statistics in the Stats dialog window. There are six machines and one fixturing station in the example model. The machine utilization report and time proportions in each machine state is provided in Figure 6.5.



The screenshot shows a window titled ".AgvModel.Stats.machine_stats" with a menu bar (File, Edit, Format, Extra) and a "Formula" checkbox. Below is a table with 8 columns: string 1, string 2, real 3, real 4, real 5, real 6, real 7, and real 8. The table contains 7 rows of data, with the first row highlighted in pink. The rows represent different machine types and their utilization statistics.

string 0	string 1	string 2	real 3	real 4	real 5	real 6	real 7	real 8
0	machine_type	machine_no	broken	idle	operating	setup	blocked	total
1	fixturing	1	0.00	10015.38	78320.00	0.00	1064.62	89400.00
2	machine	1	0.00	18441.36	70958.64	0.00	0.00	89400.00
3	machine	4	0.00	29941.06	59458.94	0.00	0.00	89400.00
4	machine	5	0.00	34815.65	54584.35	0.00	0.00	89400.00
5	machine	6	0.00	43408.28	45991.72	0.00	0.00	89400.00
6	machine	3	0.00	24422.51	64977.49	0.00	0.00	89400.00
	machine	2	0.00	23195.11	66204.89	0.00	0.00	89400.00

Figure 6.5. The final machine utilization report

The fixturing station is a potential bottleneck in the overall system. Its total operating time is reported as 78320 seconds, which yields a machine utilization of 87%. It also has a blockage time of 1064 seconds because of the capacity limit of the fixturing space in front of it.

Machine utilization of other machines can be calculated by dividing the total operating time of the machine by the total simulation time, 89400 seconds. The utilization of the six machines are 79%, 74%, 73%, 66%, 63%, and 61%. The utilization values are consistent with the model because, according to the machine allocation rules applied in the system, the lower numbered machines are utilized more frequently than the higher numbered machines.

6.2.4. Zone Statistics

The zone statistics reports provided in SIMPLE++ AGV library are consistent with the results given in Pritsker's SLAM model. In this section, first, the total number of vehicle passes from each segment of the model will be presented and compared with the segment statistics of the SLAM model. Second, the collection of blockage statistics on segments will be illustrated.

The total number of vehicle passes from each segment of the layout can be observed by selecting the specific zone object and requiring the statistics for that object. For example, Figure 6.6. shows an example zone statistics output for a specific zone object.

string 0	string 1	integer 2	integer 3	integer 4	integer 5	integer 6
	Name	Entries	Exits	Contents	PreCont	Max_Cont
1	agvs_straight15					
2	-trackEntr	524	524	0	0	1
3						
4						

Figure 6.6. A sample zone statistics report

Table 6.2. is the list of total number of passes from each segment of the path. The results from the SLAM model have also been provided in this table. As the comparison shows that there is not a big difference between the results obtained in SLAM and SIMPLE++.

Table 6.2. Total number of passes from each segment

FROM	TO	SIMPLE++	SLAM
fixturing	Intersection-1	368	365
Intersection-1	Intersection-2	523	528
Intersection-2	Parking	270	285
Parking	Intersection-3	270	284
Intersection-2	Intersection-3	253	244
Intersection-3	Intersection-4	524	528
Intersection-4	Machine-1	357	354
Machine-1	Machine-2	357	354
Machine-2	Machine-3	357	354
Machine-3	Intersection-5	357	354
Intersection-4	Machine-4	167	174
Machine-4	Machine-5	167	174
Machine-5	Machine-6	167	174
Machine-6	Intersection-5	167	174
Intersection-5	Intersection-1	524	528

The blockage information on each zone can also be obtained in SIMPLE++ AGV simulator. Figure 6.7 is a sample statistics report showing the number of total visits to a zone and the total number of blockages occurred in that zone. It is also possible to see the maximum blocking time occurred in a specific zone object.

Other zone statistics reports provided in the SIMPLE++ AGV simulator include the total number of vehicle passes from a zone, the total number of loads and unloads at workstations, and the total number of vehicle passes at intersection points. Table 6.3. shows the total number of vehicle passes, and the total number of loads and unloads at six machines and at the fixturing station.

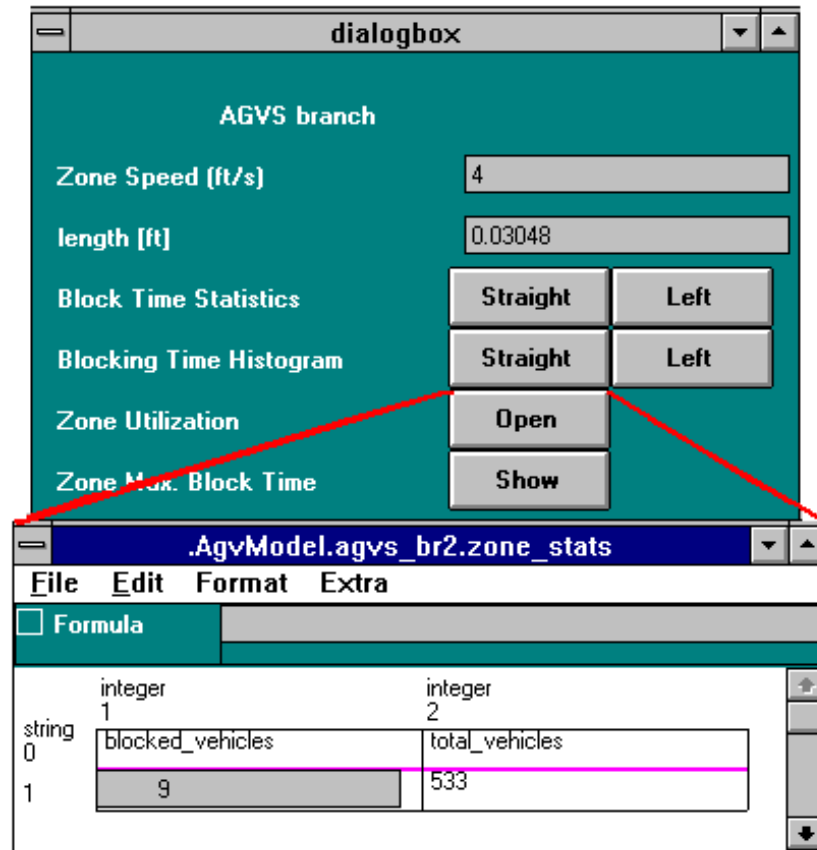


Figure 6.7. A sample report showing the number of blockages in a zone

According to this table, the observed number of vehicle passes at stations are close to the results reported in the SLAM model. The total number of loads at a machine can be one unit less than the total number of unloads. This means that there is a part in the station not yet loaded to a vehicle.

Table 6.3. Total number of vehicle entries, loads, and unloads at stations

Machine	Number of Loads	Number of Unloads	Total Entry (SIMPLE++)	Total Entry (SLAM)
Fixturing	191	196	368	365
Machine 1	38	39	357	354
Machine 2	36	36	357	354
Machine 3	34	35	357	354
Machine 4	30	31	167	174
Machine 5	29	30	167	174
Machine 6	24	24	167	174

The total number of vehicle passes at intersection points is reported in Table 6.4. The corresponding results from the SLAM model have also been provided in the table. The comparison of the results yield that there is no reason to consider the output from the SIMPLE++ AGV simulator as unacceptable. Moreover, the reports are more detailed in the proposed simulator. Table 6.4. not only shows the total number of vehicle passes on the intersections, but also provides the number of passes on each side of an intersection. For example, at Intersection-2, there were 523 vehicle passes. 253 of them were moved towards the straight side of the intersection. 270 of them were moved towards the other side of the intersection.

Table 6.4. Total number of vehicle passes at intersections

Intersection	Passes on Straight side	Passes on Second side	Total passes (SIMPLE++)	Total passes (SLAM)
Intersection-1	368	735	891	893
Intersection-2	253	270	523	528
Intersection-3	271	253	524	528
Intersection-4	167	357	524	528
Intersection-5	357	167	524	528

6.3. Validation of the Capabilities of the Simulator

During the simulation of the example model, some capabilities of the generic AGV simulator have already been demonstrated. In this section, first, these already demonstrated capabilities will be reviewed briefly, and then the rest of the capabilities of the proposed simulator, which were not shown in the example model, will be illustrated with some additions to the current model.

The original model utilizes bi-directional flow on the special spur segment. The zone objects and the dock station object on this spur segment have been defined as bi-directional (by setting the attribute uni-directional to “false”). This has been a good example of specifying a single section of the AGV path as bi-directional. There can be at most one vehicle in this spur segment. The simulator also provides a rule at the `agvs_tl_br` (terminal line branch) object in order to prevent the entry of another vehicle into the spur while it is occupied.

One of the job request types has already been illustrated in the example model. In this type, the jobs arrive at the system from a specific station and at specified arrival times. The time between arrivals for a specific part type is set by the user. Upon entering the system, the job follows the operations plan and it is routed to the workstations according to its operations sequence. If one or more machines are available, the job is allocated to a machine (based on the lower-numbered-machine-first rule). Machines and the fixturing station have a total capacity of one unit. All of these requirements have been easily programmed by using the proposed simulator.

The concept of independent workstations has also been used in the example model. The fixturing station is an example independent station. It does not directly interact with the AGV path. However, the machining times at the fixturing station affects the simulation results.

Priority assignments have also been shown in the example model. The processed jobs have priority over the first arriving jobs at the fixturing station. This is accomplished by setting the unloading operation at the fixturing station as a high priority operation. As a result, the vehicles are assigned to the processed jobs first and the jobs waiting in the queue in front of the fixturing station are ranked according to their priority status.

These are the capabilities that have already been shown in the example model. In the following sections, the other capabilities of the simulator will be demonstrated with some additions to the current model.

6.3.1. Multiple-Load Carrying Capability

Many AGV systems utilize vehicles that can carry multiple loads. Therefore, the generic AGV simulator must be capable of simulating multiple-load carrying vehicles. The example model has been revised for the purpose of validating this capability of the simulator.

The load capacity of an AGV can be specified in the dialog window of the AGV object. For demonstration purposes, the capacity of the AGVs in the example model have been set to two units. The total capacity of the fixturing station has been set to three units in order to increase the amount of parts that are requesting a vehicle at the fixturing station. Now, the vehicles pick up two units at the fixturing station if there are two or more units waiting at the fixturing space. The loads are destined for different machines, because each machine can only process one unit at a time. The vehicle is freed after unloading both units.

6.3.2. Battery Charging

In the example model, the battery constraints were not considered. The simulation results can be more reliable by applying the battery constraints. In order to simulate the model with battery charging features, the user must select the battery charging technique to be used.

A `bat_charge_station` object has been added to the current model. In the dialog window of the AGV object, the battery charging technique has been set to normal charging technique, and the values of the attributes that are related to battery charging (e.g. battery safety level, current battery charge, maximum battery capacity, battery charging current, battery consumption rate) have been identified. In this new configuration, the vehicles are sent to the battery charging station when the battery charge level reaches to the specified safety level (see Figure 6.8.)

6.3.3. Vehicle Breakdowns

AGVs are subject to breakdowns and minor delays. The generic AGV simulator is capable of simulating these breakdowns and minor delays. The example model has been revised and a Maintenance object has been placed next to the `bat_charge_station` in order to demonstrate this capability. Figure 6.9. shows the new shape of the sample AGV model layout.

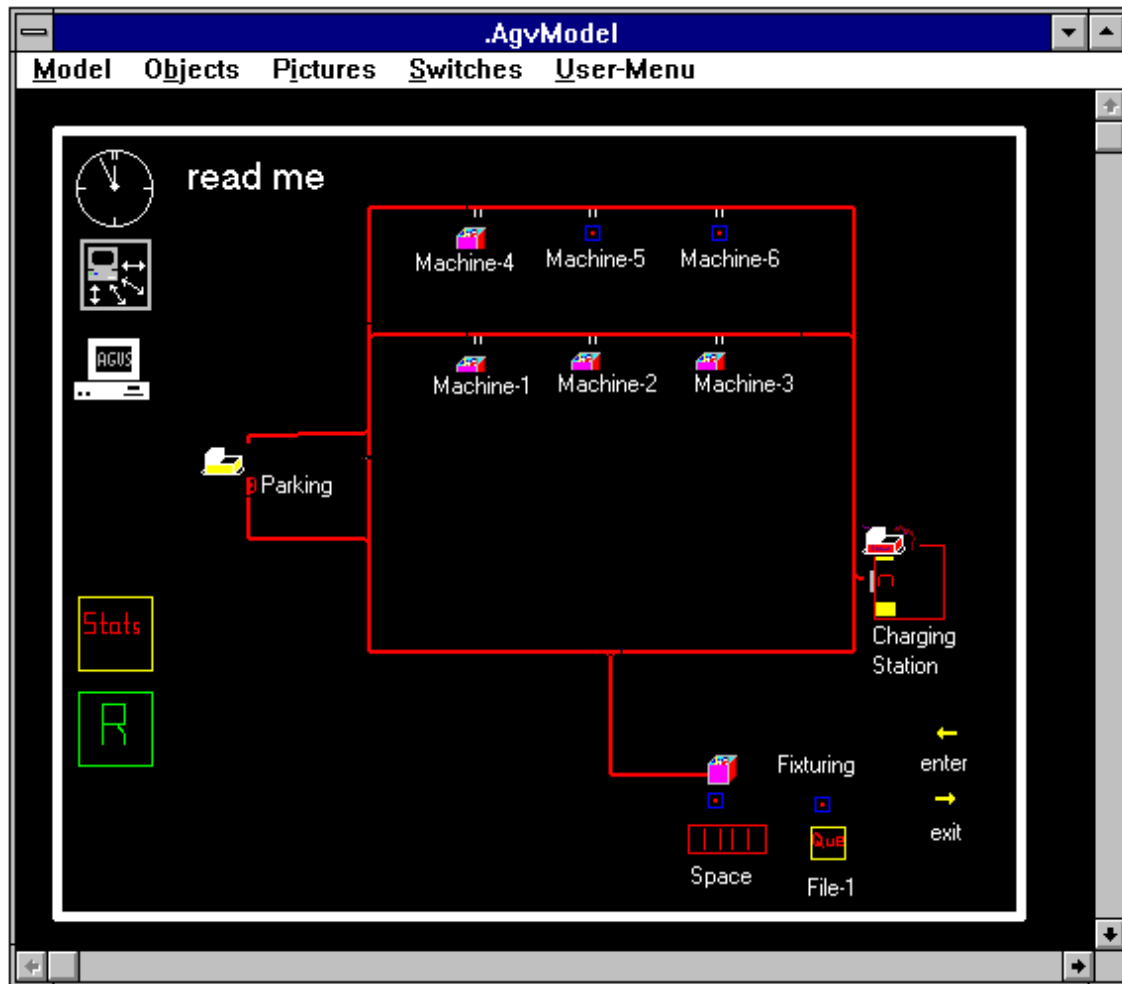


Figure 6.8. The example model with a new battery charging station

The breakdown switch in the dialog window of the AGV object must be turned on in order to activate the vehicle breakdowns. Then, the time between two consecutive breakdowns and the repair time are set in this dialog window. These times are defined as custom attributes of the AGV object (see Figure 6.10.) For example, the attribute named `break_interval` defines the time between two consecutive vehicle breakdowns. The current setting for this attribute is constant and 20 minutes. However, it can be defined as a random distribution, as well. Similarly, the attribute named `repairTime` is used for the definition of the repair time of a vehicle.

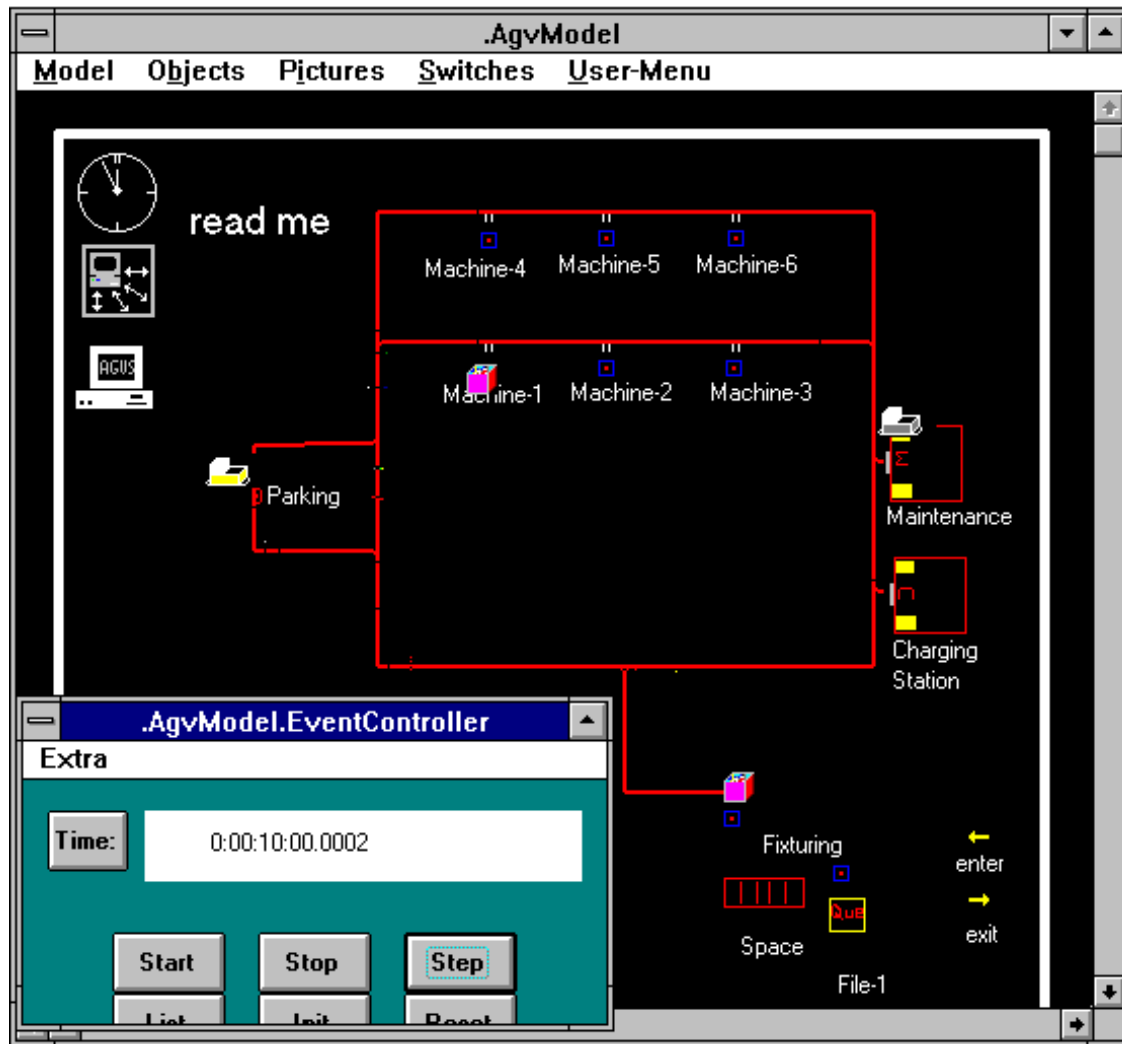


Figure 6.9. The new configuration of the model with the maintenance object

The minor delays can also be simulated by activating the type B breakdowns in the example model. A type B breakdown is a temporary delay in which the vehicle is not taken off the path. In the agvs_manager object dialog window, there is a button for setting the general specifications of an AGV model. By pressing this button, the user selects the appropriate options and defines the percentage of the occurrence of type B breakdowns (see Figure 6.11.)

The screenshot shows a dialog window titled ".agv:1" with a "Show" button. The window contains several input fields and checkboxes for configuring an AGV. The "Name" field is set to "agv". The "Nr." field is set to "00001". There are checkboxes for "Disrupted" and "Pause", both of which are unchecked. The "Basic cons." field is set to "[A] 0" and the "Driving cons." field is set to "[A] 100". Below these fields are three buttons: "insert", "edit", and "delete". A large text area contains a list of custom attributes, with "break_on=True (boolean) ni" selected. The attributes listed are: shiftWork=False (boolean) i, loadTime=45.0000 (const) (time) i, break_on=True (boolean) ni, break_interval=20:00.0000 (const) (time) ni, repairTime=2:00.0000 (const) (time) ni, fd_idleStatus=parking (string) ni, IdleMovingTime=0.0000 (const) (time) ni, transport_no=0 (integer) ni, vehicle_type=tow (string) ni, repairStayTime=1:00.0000 (const) (time) ni, origin_station=? (object) i, unloadTime=45.0000 (const) (time) i, loaded_speed=4 (speed) i, normal_speed=4.5 (speed) i, charge_control=False (boolean) i, and charge_con_interval=2:00:00.0000 (const) (time) i. At the bottom of the dialog are three buttons: "OK", "Apply", and "Cancel".

Name ☒ **Nr.:** 00001 ☐ **Disrupted** ☐ **Pause**

Basic cons. [A] ☒ **Driving cons.** [A] ☒

Cust. attributes:

shiftWork=False (boolean) i
loadTime=45.0000 (const) (time) i
break_on=True (boolean) ni
break_interval=20:00.0000 (const) (time) ni
repairTime=2:00.0000 (const) (time) ni
fd_idleStatus=parking (string) ni
IdleMovingTime=0.0000 (const) (time) ni
transport_no=0 (integer) ni
vehicle_type=tow (string) ni
repairStayTime=1:00.0000 (const) (time) ni
origin_station=? (object) i
unloadTime=45.0000 (const) (time) i
loaded_speed=4 (speed) i
normal_speed=4.5 (speed) i
charge_control=False (boolean) i
charge_con_interval=2:00:00.0000 (const) (time) i

Figure 6.10. Definition of custom attributes for agv-1

6.3.4. Job Arrival Types

In the current model, jobs arrive at the system randomly and from the enter station. Another type of job arrival mechanism may employ deterministic values for the job arrival times. If such a mechanism is required for the example model, the user must modify the current model by applying the following steps. First, the dialog window of the enter station object is opened. In this window, the part type part1 is selected from the job requests menu.

The next appearing window is a dialog window of a Generator object which is used for setting the arrival times for part1. Here, all values must be set to zero in order to deactivate the generation of part1. Second, the user must open the dialog window of the ProdMgr object, and press the button named Order. An empty table will appear. The orders from each part type and their quantities and release dates can be written into this table as shown in Figure 6.12.

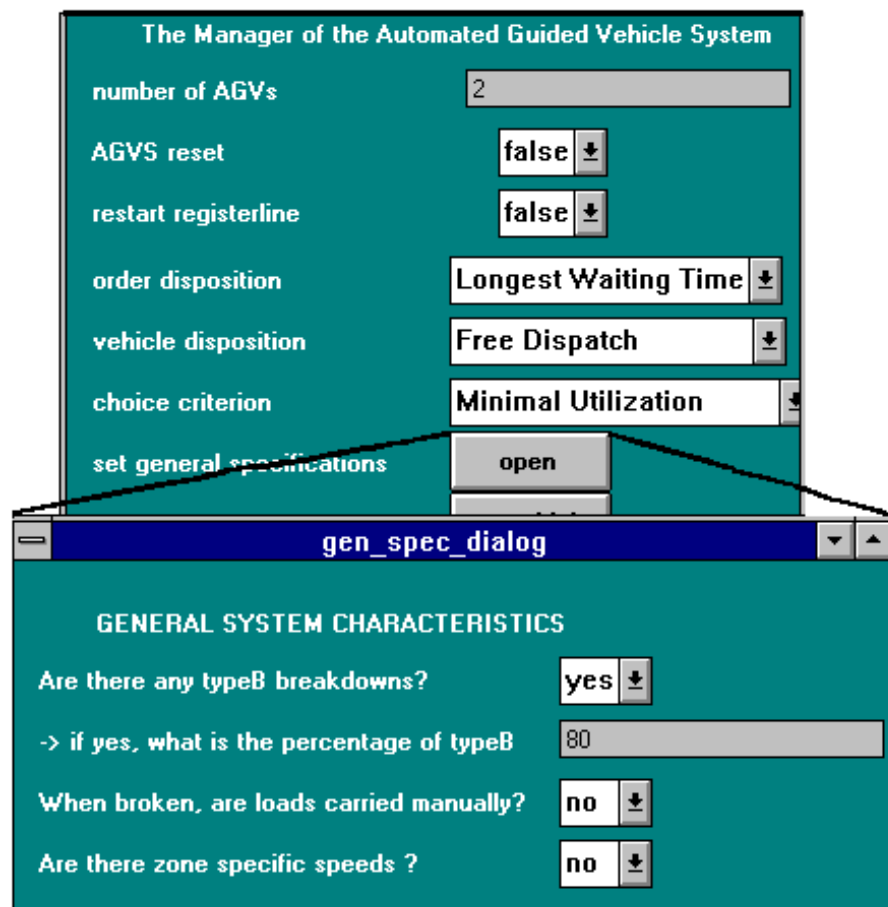


Figure 6.11. Illustration of activating type B breakdowns

	string 1	string 2	integer 3	datetime 4
string 0	orderNo	EntityType	number	release date
1	1	part1	5	
2	2	part1	5	1970/01/01 01:00:00
3				
4				

Figure 6.12. The table containing part ordering information

Now, the simulator will utilize these new information for generating parts. This type of ordering mechanism is helpful when the production plan is known and the system is going to be tested according to this plan. The simulation ends when the last order exits the system. Therefore, the user does not have to specify the ending time of the simulation.

6.3.5. Vehicle Assignments

There may be two types of vehicle assignments in the example model. First, a vehicle can be assigned to serve only for specific stations, but not for others. Second, a vehicle can be assigned to serve only for specific part types, but not for other part types. In this section, these vehicle assignment features will be demonstrated. The dialog window of a dock station object has a button called Assign. After the initialization of a model, the user can see a list of vehicles by pressing this button (see Figure 6.13.) By deleting a vehicle from this list, that particular vehicle is disabled from serving the dock station object. As an example, agv-1 has been disabled from serving the fixturing station by deleting its name from the corresponding vehicles list.

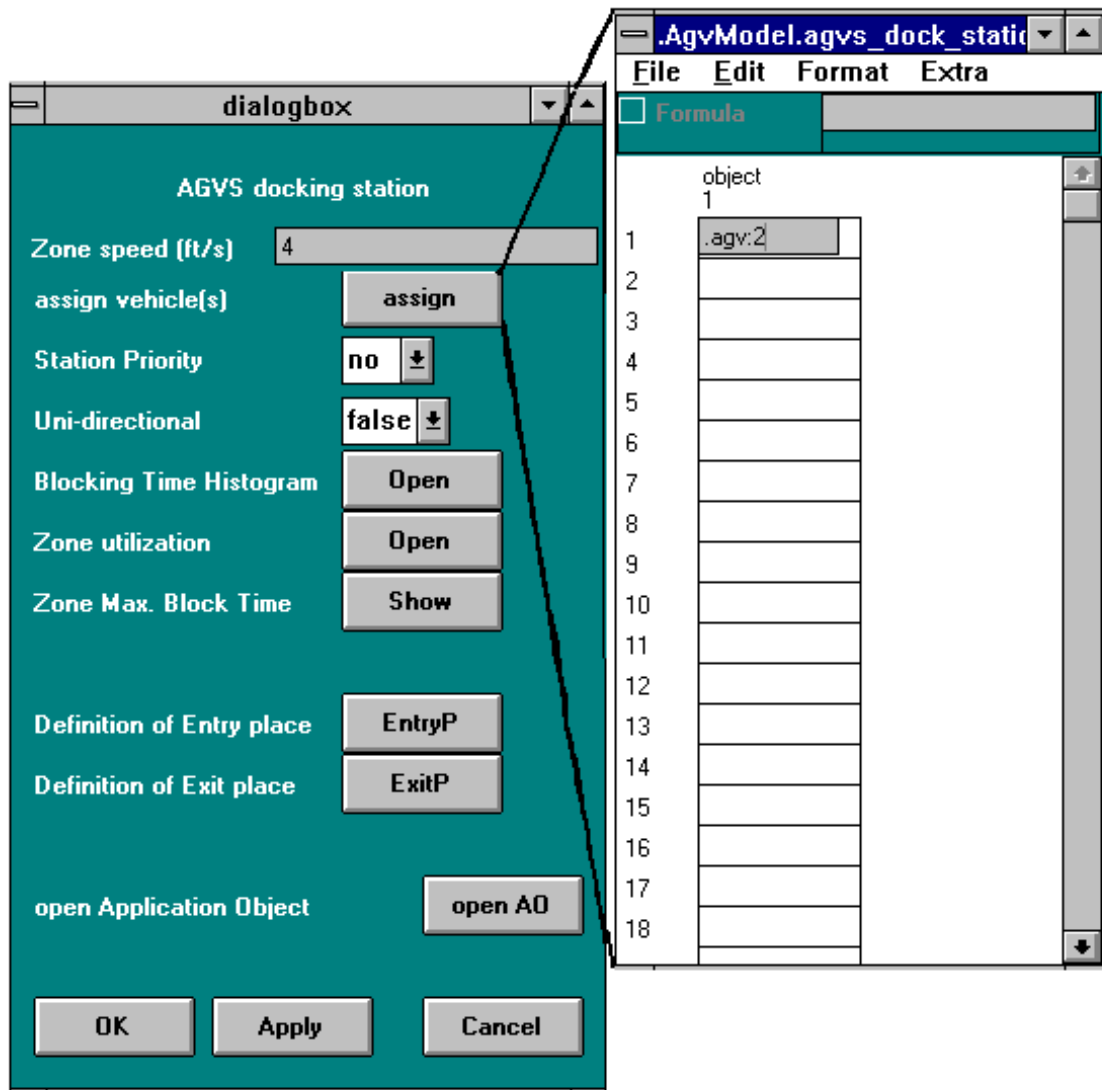


Figure 6.13. Assigning vehicles to serve a specific dock station

It is also possible to assign a vehicle to serve for specific part types. For demonstration, a second part type have been added to the model. The master plan prepared for the system can be seen by pressing the button named Master in the dialog window of the ProdMgr object. A second part type (part2) has been defined for the example model (see Figure 6.14.) After defining the new part type and its operations plan, the user must initialize the system in order to register the new part type to the ProdMgr object.

In the master plan table, the column titled as “vehicles” will contain the list of vehicles that are assigned for a part type (see Figure 6.14.) In this list, some vehicles can be deleted to disable them from serving the particular part type.

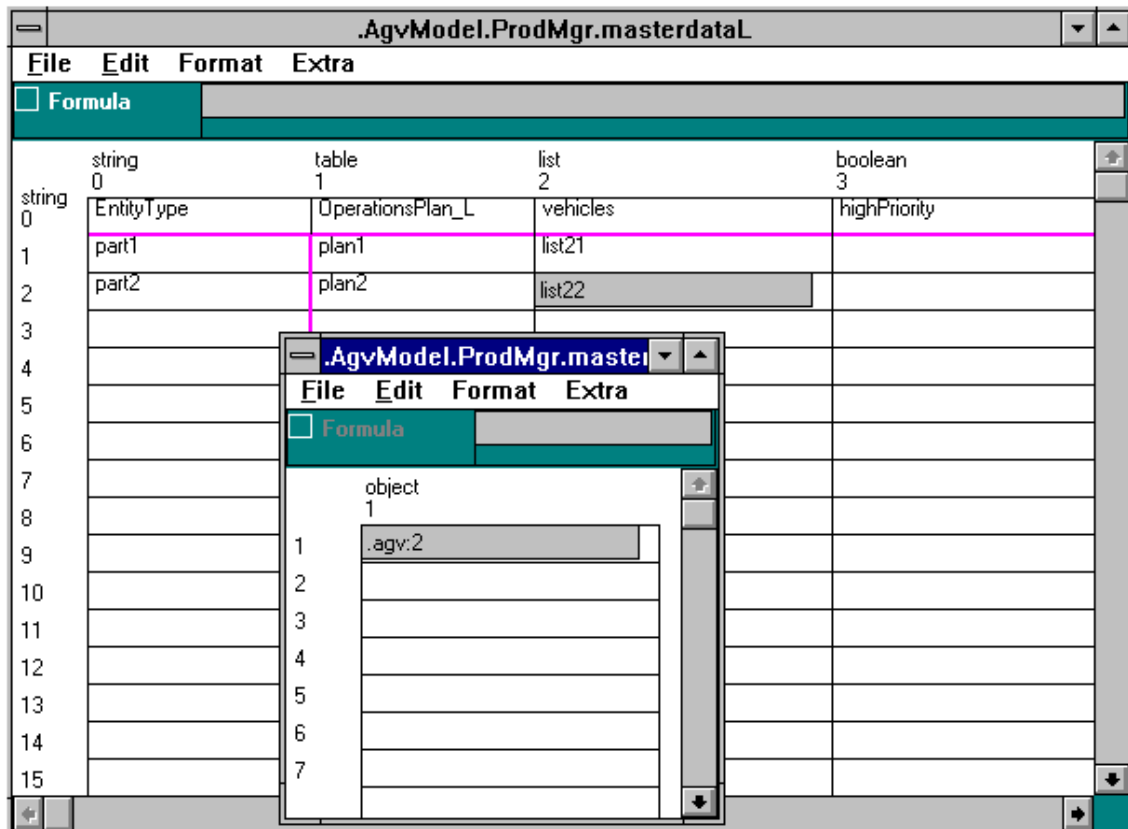


Figure 6.14. Assigning vehicles to serve specific part types

6.3.6. Traffic Rules

At intersections, there may be traffic rules that are applied to prevent congestion. In the dialog window of a junction element, a pop-up menu has been provided for selecting a specific traffic rule. There are currently four rules in this pop-up menu. Figure 6.15. shows the dialog window of a junction element that contains the menu displaying the traffic rules.

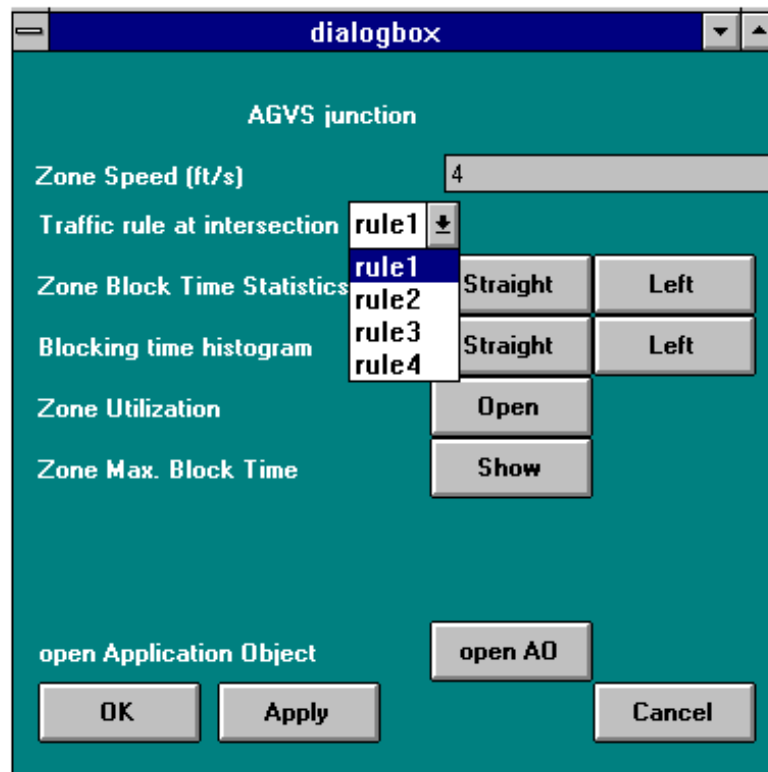


Figure 6.15. The dialog window of a junction object

The first rule is the default. It is basically the first-come-first-serve rule. The second rule gives priority to an occupied or assigned vehicle. The third and fourth rules can be used if there is a directional priority (e.g. right-side-first rule) at the junction.

6.4. Conclusion

In this chapter, the statistics abilities of the generic AGV simulator have been illustrated and the results of the simulation of the example model have been proved to be close to the results of the SLAM model. This has been the first part of the validation of the simulation program. The aim of the second part of the validation has focused on the demonstration of several capabilities of the package.

Chapter VII

Conclusions and Recommendations

7.1. Conclusions

In this study, a generic AGV simulator has been developed for simulating a variety of AGV systems. The simulator has been built upon the AGV library of a commercial object-oriented simulation package, SIMPLE++. The simulator is capable of reading the layout information from a CAD drawing provided that the drawing is modified for the data translation.

The modifications made to the old AGV library include both adding new objects and changing the structure of the existing objects. The new AGV application objects that have been added to the library include AGV Model Frame, Stats, RouteMgr, Battery Charger, Maintenance, and Build. Some of the existing objects in the old library have been modified in order to support the desired capabilities of the generic AGV simulator.

Two example models have been developed for model validation and verification purposes. The first example model (Pritsker's SLAM model) has been chosen from the literature and the corresponding SIMPLE++ model has been developed and tested by the simulator. The results obtained in this study were consistent with the results reported in the past model. The capabilities of the generic AGV simulator have also been demonstrated by modifying the example model. The second example model is a larger AGV model which contains 22 workstations, five part types and five vehicles.

The built-in language of the SIMPLE++ simulation package is a powerful language. The author did not need to use another programming language (e.g. C++) for programming the routines for data translation from a CAD environment. The object-oriented structure of SIMPLE++ allows modular programming, and thus, the package is flexible and reusable. However, the AGV library of the SIMPLE++ is not as flexible as SIMPLE++ itself. For example, the original AGV library was not capable of simulating multiple-load carrying vehicles. The library has been modified in order to achieve more flexibility in simulating AGV systems.

7.2. Recommendations

Collecting statistics from the simulator can take a long time if the user does not consider the memory constraints of the computer. For this reason, the statistics options have been categorized into 5 sections (job statistics, vehicle statistics, machine statistics, buffer statistics, and zone statistics.) The user can activate/deactivate each option during the simulation.

It is suggested to reload a model after a long run if the speed of the computer is not good enough. After a long run with statistics options on, it may take a long time to reset the program and start another simulation. It takes less time to reload the program and start it again.

The internal language of the SIMPLE++ AGV library has some comments and some function names written in German. These comments were not critical in this study, however, they should be translated to English. During this study, many new methods have been added to the AGV library. These methods have been furnished with necessary explanations and comments.

One restriction of the simulator is that there may be only vertical and horizontal lines in the layout of an AGV model. This restriction of the simulator results in less flexibility in defining the AGV layout. However, a line can be approximated to fit into a vertical or a horizontal shape. As long as the length of the line is correct, the shape of the line does not affect the simulation results.

The proposed simulator can be expanded to include other material handling objects, parallel processors, and assembly operations, in the future. Furthermore, the simulator can be used as a module of an integrated manufacturing system. For example, production plan data can be input to the simulator from a production planner module and the output of the simulation can be used as input to a knowledge-based decision support system.

Bibliography

- Andersson, M. "AGV system simulation - a planning tool for AGV route layout," *Proceedings of the 3rd International Conference on AGV systems*, pp. 291-296, 1985.
- Araki, T., Takahashi, T., Suekane, M., and Kawai, M. "Flexible AGV system simulator," *Proceedings of the 5th International Conference on AGV Systems*, pp. 77-86, 1987.
- Blair, E.L. and Selvaraj, S. "DISC++: A C++ based library for object-oriented simulation," *Proceedings of the 1989 Winter Simulation Conference*, pp. 301-305, 1989.
- Bozer, Y.A. and Srinivasan, M.M. "Tandem configurations for AGVS and the analysis of single vehicle loops," *IIE Transactions*, v. 23, no.1, pp. 72-82, 1991.
- Braizer, M.K. and Shannon, R.E. "Automatic programming of AGVS simulation models," *Proceedings of the 1987 Winter Simulation Conference*, pp. 703-708, 1987.
- Choi, H., Kwan, H., and Lee, J. "Traditional and tandem AGV system layouts: A simulation study," *Simulation*, v.63, no. 2, pp. 85-93, 1994.
- CMS Research, MAST Simulation Environment - User's Manual Version 3.0, 1990.
- Dewsnup, M. C. "How to model AGVS using ProModel for Windows," *Proceedings of the 1995 Winter Simulation Conference*, pp. 482-486, 1995.

- Dutt, S. "Guided vehicle systems - A simulation analysis," M.Sc. Thesis, Department of Industrial and Systems Engineering, Virginia Polytechnic Institute and State University, 1991.
- Egbelu, P.J. "The use of non-simulation approaches in estimating vehicle requirements in an AGV based transport system," *Material Flow*, v. 4, pp. 17-32, 1987.
- Egbelu, P.J. and Tanchoco, J.M.A. "AGVSim User's Manual," Technical Report No. 8204, Department of Industrial Engineering and Operations Research, Virginia Polytechnic Institute and State University, Blacksburg, VA, 1982.
- Ethier, S. J. and Ethier, C.A. "AutoCAD for Success," Prentice Hall, Englewood Cliffs, New Jersey, 1996.
- Feare, T. "Simulating materials handling systems," *Modern Materials Handling*, v.51, no. 3, pp. 39-41, March 1996.
- Fishwick, P.A. "SIMPACK: Getting started with simulation programming in C and C++," *Proceedings of the 1992 Winter Simulation Conference*, pp. 154-162, 1992.
- Gaskins, R.J. and Tanchoco, J.M.A. "AGVSim2 - a development tool for AGVS controller design," *International Journal of Production Research*, v. 27, no. 6, pp. 915-926, 1989.
- Gong, D. and McGinnis, L.F. "An AGVS simulation code generator for manufacturing applications," *Proceedings of the 1990 Winter Simulation Conference*, pp. 676-682, 1990.
- Gould, L. "AGVs in America: An inside look," *Modern Materials Handling*, vol. 45, no. 10, pp. 56-60, September, 1990.

- Gould, L. "Selecting an AGVS: new trends, new designs," *Modern Materials Handling*, vol. 50, no. 6, pp. 42-43, May, 1995.
- Govindaraj, T., McGinnis, L.F., Mitchell, C.M., Bodner, D.A., Narayanan, S., and Sreekanth, U. "OOSIM: A tool for simulating modern manufacturing systems," *Proceedings of the 1993 Design and Manufacturing Systems Grantees Conference*, pp. 1055-1062, 1993.
- Hammond, G. "AGVs at work," IFS Publications Ltd., UK., 1986.
- Henriksen, J. and Crane, R. "GPSS/H Reference Manual," Annandale, VA, Wolverine Software Corporation, 1989.
- Jayaraman, A. "Use of simulation-animation techniques in the design of an AGV system," M.Sc. Thesis, Department of Industrial and Systems Engineering, Virginia Polytechnic Institute and State University, 1993.
- Johnson, M.E. and Brandeau, M.L. "An analytical model for design of a multivehicle automated guided vehicle system," *Management Science*, v. 39, no. 12, p. 1477, December, 1993.
- Joines, J.A., Powell, K.A., and Roberts, S.D. "Object-oriented modeling and simulation," *Proceedings of the 1992 Winter Simulation Conference*, pp. 145-153, 1992.
- King, R.E. and Kim, K.S. "AgvTalk" An object-oriented simulator for AGV systems," *Computers and Industrial Engineering*, v. 28, no. 3, pp. 575-592, 1995.
- Klein, C.M. and Kim, J. "AGV dispatching," *International Journal of Production Research*, v. 34, no. 1, pp. 95-110, 1996.

- Koff, G.A. "Automatic Guided Vehicle Systems : Applications, Controls and Planning," *Material Flow*, vol. 4, pp. 3-16, 1987.
- Lalonde, W.R. and Pugh, J.R. "Inside Smalltalk," vol. 2, Prentice-Hall, Englewood Cliffs, NJ, 1990.
- Law, A.M. and Kelton, W.D. "*Simulation Modeling and Analysis* ," McGraw Hill, New York, 1991.
- Lee, J. "Composite dispatching rules for multiple-vehicle AGV systems," *Simulation*, v.66, no. 2, pp. 121-130, 1996.
- Lee, J., Choi, R.H., and Khaksar, M. "Evaluation of AGVS by simulation," *Computers and Industrial Engineering*, v. 19, nos. 1-4, pp. 318-321, 1990.
- Lomow, G. and Baezner, D. "A tutorial introduction to object-oriented simulation and Sim++," *Proceedings of the 1990 Winter Simulation Conference*, pp. 149-153, 1990.
- Mahadevan, B. and Narendran, T.T. "Estimation of number of AGVs for an FMS: an analytical model," *International Journal of Production Research*, v. 31, no. 7, pp. 1655-1670, 1993.
- Marsh, R. J. "Extend Simulation Software," APICS-The Performance Advantage, Software Review, January, 1995.
- Maxwell, W.L. and Muckstadt, J.A. "Design of AGVS," *IIE Transactions*, v. 14, no. 2, pp. 114-121, 1982.
- Material Handling Institute, "AGVS Application Profiles," The Material Handling Institute, Charlotte, NC, USA, 1993.

- McHaney, R. "Modeling battery constraints in discrete event AGV simulations," *International Journal of Production Research*, v. 33, no. 11, pp. 3023-3040, 1995.
- Miller, R.K. "Automated Guided Vehicle Systems," SME Publications, Dearborn, MI, 1987.
- Mosca, R., Giribone, P., and Schenone, M. "O.R. simulation model referring to AGV carriages transportation network in a large dimensions fruit and vegetable market," *International Journal of Modelling and Simulation*, v.11, no. 3, pp. 104-109, 1991.
- Narayanan, S., Bodner, D.A., Sreekanth, U., Govindaraj, T., McGinnis, L.F., and Mitchell, C.M. "Research in object-oriented manufacturing simulations: An assesment of the state of the art," Working Paper submitted to IIE Transactions, August 30, 1995.
- Norman, V.B., Norman T.A., and Fornsworth, K. "Rule-based simulation of AGV systems," *Proceedings of the 5th International Conference on AGV Systems*, pp. 113-120, 1987.
- Ozden, M. "A simulation study of multiple-load-carrying AGVs in a FMS" *International Journal of Production Research*, v.26, no.8, pp. 1353-1366, 1988.
- Pegden, C.D., Shannon, R.E., and Sadowski, R.P. "Introduction to simulation using SIMAN," New York, McGraw-Hill, Inc., 1990.
- Prasad, K. and Rangaswami, M. "Analysis of different AGV control systems in an integrated IC manufacturing facility, using computer simulation," *Proceedings of the 1988 Winter Simulation Conference*, pp. 568-574, 1988.

Pritsker, A.A.B. "Introduction to simulation and SLAM II," 4th ed., New York: John Willey & Sons, 1995.

Quinn, E.B. "A simulation based system for automatic development and testing of an AGV control software," *Proceedings of the 3rd International Conference on AGVS*, pp. 219-228, 1985.

Schulze, L. and Rosenbach, K.D. "Computer applications for the planning of AGVS," *Proceedings of the 5th International Conference on AGVS*, pp. 87-101, 1987.

Schwetman, H.D. "Introduction to process-oriented simulation and CSIM," *Proceedings of the 1990 Winter Simulation Conference*, pp. 154-157, 1990.

Seifert, R.W., Kay, M.G., and Wilson, J.R. "Evaluation of AGV routing strategies using hierarchical simulation," *Proceedings of the 1995 Winter Simulation Conference*, pp. 850-856, 1995.

Shen, Y. "The impact of vehicle dispatching on the design of multiple-transporters material handling systems," Ph. D. Dissertation, Department of Industrial and Systems Engineering, Virginia Polytechnic Institute & State University, 1995.

Sinriech, D. and Tanchoco, J.M.A. "Impact of empty vehicle flow on performance of single-loop AGV systems," *International Journal of Production Research*, v.30, no. 10, pp. 2237-2252, 1992.

Sreekanth, U. "POEMS - A Process and Object Environment for Manufacturing Simulations," M. Sc. Thesis, School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA, 1994.

- Srinivasan, M.M., Bozer, Y.A., and Cho, M. "Trip-based material handling systems: Throughput capacity analysis," *IIE Transactions*, v. 26, no. 1, p. 70, January, 1994.
- Taghaboni-Dutta, F. and Tanchoco, J.M.A. "Comparison of dynamic routeing techniques for automated guided vehicle systems," *International Journal of Production Research*, v.33, no. 10, pp. 2653-2669, 1995.
- Taha, H.A. "Simulation with SIMNET II," SimTech, Fayetteville, Arkansas, 1992.
- Takakuwa, S. "Design and cost-effectiveness analysis of large-scale AS/RS-AGV systems," *Proceedings of the 1993 Winter Simulation Conference*, pp. 851-856, 1993.
- Tanchoco, J.M.A. "*Material Flow Systems in Manufacturing* ," Chapman&Hall, 1994.
- Tanchoco, J. M. A. and Co, C. G. "Real-time Control Strategies for Multiple-load AGVs," Research Memorandum No. 90-14, School of Industrial Engineering, Purdue University, September, 1990.
- Tanchoco, J.M.A, Egbelu, P.J., and Taghaboni, F. "Determination of the total number of vehicles in an AGV-based material transport system," *Material Flow*, v. 4, pp. 33-51, 1987.
- Taylor Times Simulation Magazine, Spring, 1996.
- Ulgen, O.M. and Kedia, P. "Using simulation in design of a cellular assembly plant with automatic guided vehicles," *Proceedings of the 1990 Winter Simulation Conference*, pp. 683-691, 1990.

Warnecke, H.J., Steinhilper, R., and Zeh, K.P. "*Simulation as an integral part of FMS planning, Simulation Applications in Manufacturing*," IFS Publications, Berlin, Germany, pp. 131-147, 1986.

Wilson, J.K. "A C-based simulation framework for AGVS," M.Sc. Thesis, Department of Industrial and Systems Engineering, Virginia Polytechnic Institute and State University, 1992.

Appendix-1

TRANSFERING CAD DRAWINGS INTO SIMPLE++

AGV vendors usually provide a drawing of the AGV path to their customers in the form of a computer aided drawing. There are many packages used for drawing purposes. One of the most popular commercial CAD packages used in manufacturing systems is AutoCAD. In this study, due to its popularity and availability in manufacturing environments, AutoCAD (Release 12) has been used as a drawing tool.

Drawing aids of the AutoCAD package such as layers, blocks, attributes, and data exporting feature have been utilized in transferring the layout data into SIMPLE++. These concepts will be briefly introduced in the following paragraphs, but the reader must refer to an AutoCAD reference for more information.

Layers are used to group entities in a drawing into separate categories so that they may be displayed on the screen in a variety of combinations. The layers can be displayed singly or collectively. The layout drawing of an AGV path can be categorized into five layers. These layers are: path, control points, intersections, pick up/deposit points, and directions. The path layer includes only lines and curves. This layer is used for the raw drawing which does not contain any information for the simulator. The user is responsible for drawing the other entities (control points, directions, etc.) onto the raw drawing so that the output data from the CAD drawing can be meaningful to the simulation program.

When a part of a drawing is used frequently, blocks can be used. A block is created as a file and stored in an internal block library. The block is called from the library when it is needed. This allows the drawing to go anywhere and still have its blocks travel with it. In this study, the blocks have

been used for representing the intersection elements (branches and junctions), arrows showing the directions (up, down, left, right), and pickup/deposit points. A special blocks library have been created for the purpose of preparing CAD drawings to be transferred to SIMPLE++. The blocks are inserted onto the raw drawing manually. There is a tolerance value for this insertion process. Even if the coordinates of a line and a block do not match exactly, the placement may be accepted if the difference between the coordinates is not greater than the tolerance value.

Attributes are storage registers for text information that travels with blocks. The attribute concept has been utilized in this research study in order to define the type of an intersection or a pick up/deposit point. For example, when a block representing an intersection is inserted, the type of the intersection is asked. The name of an intersection type must be correctly typed by the simulation programmer (see Figure A.3. for the names of intersection types.)

A drawing exchange file (DXF) format has been provided in order to make the CAD drawing data available outside the AutoCAD program. The DXF file is a standard ASCII text file that lists all the data about a drawing. To export the drawing data in the DXF format, the DXFOUT command must be used. A file with the extension “dxf” is created by using this command. The .dxf file is used to transfer the data into SIMPLE++.

Preparing the CAD Drawing for Data Transfer

The initial drawing of an AGV path contains only line and curve entities. These entities are drawn on the path layer. All other entities that are necessary for the transfer process must be inserted onto the initial drawing, but at different layers. The additional entities are control points, pick up and deposit points, intersections, and directions.

The layout of the example model given in chapter 5 will be used for the illustration of preparing a CAD drawing for data transfer. The initial drawing of the example model, which contains only lines and curves, is shown in Figure A.1. In the following paragraphs, the steps for preparing the drawing before the data transfer will be described.

(Figure A.1. is going to be placed here)

First, the intersection entities are inserted on the initial drawing. Before starting this insertion process, the intersections layer must be set as the active layer. On the upper left corner of the screen, a pop-up menu displays the active layer of the drawing. The layer “INTERSECTIONS” must be selected in this pop-up menu. By using the **insert** command, the intersection entities can be inserted. The new appearance of the drawing after the insertion of intersection entities is shown in Figure A.2. During the execution of the insert command, the AutoCAD program asks for the name of the block to be inserted. The block name must be specified as “traffic” for an intersection entity. Later, the program asks for the insertion point and the orientation of the block. The insertion point must be selected by the user. For the orientation, all questions must be answered as the default values (by pressing Return /Enter.) Finally, the type of the intersection entity must be specified. There are 25 types of intersection entities that have been defined in the modified AGV library. A complete list of the intersection types and their names are given in Figure A.3.

(Figure A.2. is going to be placed here)

(Figure A.3. is going to be placed here)

Second, the control point entities are inserted. Similar to the intersections, the layer “CPOINTS” must be set as the active layer. The block name must be typed as “cp” to insert a control point entity. The shape of the drawing after this insertion process is shown in Figure A.4.

(Figure A.4. is going to be placed here)

Third, the pickup/deposit point entities are inserted. The active layer must be set to “PD” before the insertion. The block name must be typed as “pd” to insert a pick up/deposit point entity. The type of a pick up/deposit point must also be determined during the block insertion process. There are three types of pick up/deposit point entities: dock2, dock1pick, and dock1deposit. The block named dock2 is used for an interaction point at which both pickup and deposit interactions are allowed. The block named dock1pick is used for an interaction point at which only pickup interactions are implemented. Similarly, the block named dock1deposit is used for an interaction point at which only deposit interactions are allowed. The appearance of the drawing after inserting the pick up/deposit points is shown in Figure A.5.

(Figure A.5. is going to be placed here)

Finally, the direction entities are inserted on the drawing. The blocks that have been defined for representing the directions are simple arrow shapes named as: up, down, right, and left. The active layer must be set to “DIRECTIONS” before inserting these direction entities. If the direction of a segment is two-way, no direction entity is inserted on that segment. The drawing after the insertion of direction entities is shown in Figure A.6.

(Figure A.6. is going to be placed here)

Data Transfer Process

After completion of the CAD drawing, a .dxf file is created in AutoCAD by using the DXFOUT command. This file must be carried to the computer in which the SIMPLE++ software has been installed. In the AGV library of SIMPLE++, an object named Build has been developed for transferring the layout data from a CAD drawing. By a double click on this object, its dialog window appears (see

Figure A.7.) In this dialog window, the user can set the name of the .dxf file by pressing the button named Set. After setting the name of the interface file, the corresponding SIMPLE++ model is drawn automatically by pressing the first Open button in the dialog window of the Build object. This model was shown in Figure 5.2. (see Chapter V).

(Figure A.7. is going to be placed here)

Appendix II

INPUT DATA FOR THE LARGE EXAMPLE MODEL

Table B.1. Time between part arrivals for the part types in the large model

Part Type	Distribution Type	Parameters (in minutes)
Part1	Uniform	minimum = 10, maximum = 15
Part2	Exponential	average = 20
Part3	Normal	average = 30, standard dev. = 3
Part4	Triangle	average = 15, minimum = 10, maximum = 20
Part5	Uniform	minimum = 20, maximum = 25

Table B.2. Operations sequences for the part types in the large model

Part1	Part2	Part3	Part4	Part5
Entersystem	Entersystem	Entersystem	Entersystem2	Entersystem2
op4	op1	op2	op3	op6
op7	op4	op5	op8	op12
op10	op6	op16	op11	op17
op14	op9	exitsystem	op13	op18
inspection	op15		exitsystem	exitsystem
exitsystem	exitsystem			

Appendix III

NEW METHODS IN THE MODIFIED AGV LIBRARY

During this study, new objects have been developed and the existing objects in the current AGV library have also been modified. In the following, the modifications made to the existing objects and the new methods defined in the new objects are listed and explained.

New methods in the existing objects

New methods have been inserted into the existing AGV library objects including AGV System Manager, agvs_straight, agvs_dock_station, and station. In the following, these new methods are listed and their functions are explained.

Method Name: Battery_Charge

Method's parent object: AGV System Manager

Description: This method is called when the battery charge reaches the safety level specified by the user. The vehicle waits at the charging station while it is being charged.

Code:

```
-----  
(vehicle : object)  
is  
  local elapseTime : time;  
      chargeRequirement : real;  
      part, partLocation, charger : object;  
do  
if round(vehicle.batCharge) = round(vehicle.batCapacity) -- battery is full  
  then  
    vehicle.batCharging := false;  
elseif vehicle.fd_dispoStatus = "occupied" -- vehicle was occupied  
  then -- vehicle was occupied  
    vehicle.batCharge := vehicle.batCharge + 0.05;
```

```

    if vehicle.charge_control
    then
        ref(BatteryCharge2).methCall(10, vehicle);
    end;
elseif (round(vehicle.batCharge) = round(vehicle.batReserve))
    OR
    vehicle.charge_control -- vehicle is controlled periodically
    -- battery empty or reserve level reached
then
    vehicle.batCharging := true;
    charger := root.bat_charge_station.battery_charge;
    vehicle.move(charger);
    --delete vehicle from the available vehicle list
    vd_freeVehl.cursorX := 1;
    vd_freeVehl.cursorY := 1;
    if vd_freeVehl.find(vehicle)
    then
        vd_freeVehl.cutRow(vd_freeVehl.cursorY);
    end;
    if vehicle.fd_dispoStatus = "assigned"
    then -- vehicle was assigned
        -- place back the part to the waiting list
        partLocation := vehicle.destination;
        part := partLocation.partReady.mu(1);
        vd_transOrder(part, partLocation, partLocation.vehiclesL);
    end; -- end if the vehicle was assigned
    -- if the vehicle was at parking station statistics are updated separately
    if vehicle.fd_idlestatus = "parking"
    then
        root.stats.setVehIdleTimes(root.eventcontroller.SimTime -
                                   vehicle.TimeInStation,
                                   vehicle, "parking")
    end;
    -- change the status of the vehicle on tables
    vd_changeStatus(vehicle, "charging");
    vd_tableUpdate(vehicle, "newTime"); -- newT t must be capital like this
    -- check the battery filling technique
    if vehicle.bat_refill_tech = "charge"
    then
        chargeRequirement := (vehicle.batCapacity) - (vehicle.BatCharge);
        elapseTime := chargeRequirement * (1 / vehicle.batChargeCurrent) * 3600;
    elseif vehicle.bat_refill_tech = "change"
    then
        elapseTime := vehicle.bat_change_time;
    end; -- end of checking the battery filling technique
    --print elapseTime;

```

```

    ref(freeVehicle).methCall(elapseTime, vehicle);
    -- if vehicle is periodically controlled then
    -- call for the next control time
    if vehicle.charge_control
    then
        ref(BatteryCharge2).methCall(vehicle.charge_con_interval, vehicle);
    end;
end; -- end for all if's
end; -- end of the method

```

Method Name: VehicleBreakSet

Method's parent object: AGV System Manager

Description: This method is called when a vehicle is broken. The vehicle is either taken out from the path or stays on the path for the repair time.

Code:

```

(vehicle : object)
is
    local part, partLocation : object;
        randomNumber : real;
        typeB : boolean;
        status : string;
    do
        -- keep the current status
        status := vehicle.fd_dispoStatus;
        -- determination of the type of breakdown
        typeB := false; -- type B breakdown is initialized as false
        if broken_stay
            AND
            NOT (vehicle.fd_idlestatus = "parking")
        then -- type B vehicle breakdown is possible (where vehicle stays when it is
            broken)
                randomNumber := z_uniform(2, 1, 100); -- (stream, start, stop)
                if randomNumber < broken_stay_per
                then -- this is a type B breakdown
                    typeB := true;
                end;
            end;
        end;
    if (typeB
        AND

```

```

(vehicle.~.~.origin = .agvs_dock_station))
OR
(vehicle.~.~.tl)
then
    ref(vehicleBreakSet).methCall(10, vehicle);
    return;
end;
-- now if typeB is true some actions will
-- not be taken for vehicle breaks
if vehicle.fd_dispoStatus = "occupied"
then    -- vehicle was occupied
    if NOT typeB
    then -- the vehicle is taken out of path
        if broken_load_carry
        then
            ref(vehicleBreakSet).methCall(10, vehicle);
            return;
        else
            transfer_load(vehicle);
        end;
    end;
end;
--delete vehicle from the available vehicle list(for both typeA and typeB)
vd_freeVehl.cursorX := 1;
vd_freeVehl.cursorY := 1;
if vd_freeVehl.find(vehicle)
then
    vd_freeVehl.cutRow(vd_freeVehl.cursorY);
end;
if vehicle.fd_dispoStatus = "assigned"
AND
NOT typeB
then    -- vehicle was assigned and it is not taken out for maintenance
    -- place back the part to the waiting list(for both type A and type B)
    partLocation := vehicle.destination;
    part := partLocation.partReady.mu(1);
    vd_transOrder(part, partLocation, partLocation.vehiclesL);
end; -- end if the vehicle was assigned
-- if the vehicle was at parking station statistics are updated separately
-- this is only for type A breakdown
-- because type B breakdown is not possible at parking station
if vehicle.fd_idlestatus = "parking"
then
    root.stats.setVehIdleTimes(root.eventcontroller.SimTime -
                                vehicle.TimeInStation,
                                vehicle, "parking")

```

```

end;
-- if the vehicle was waiting atMachine statistics are updated separately
-- for both type A and type B
if vehicle.fd_idleStatus = "atMachine"
then
    root.agvs_manager.vd_dispoTable.find (vehicle);
    root.stats.setVehIdleTimes(root.eventcontroller.SimTime -
        vehicle.~.~.vehEntrTimeToSpur,
        vehicle, "atMachine")
end;
-- change the status of the vehicle on tables
-- for both type A and type B
vd_changeStatus(vehicle, "failed");
vd_tableUpdate(vehicle, "newTime");
if NOT typeB
then
    -- set the vehicle destination to station(for type A breakdown)
    if (vehicle.~.~.origin = .agvs_tl_br)
        AND
        (vehicle.~.~.agvInWay >= 1)
    then -- vehicle was in a terminal branch
        vehicle.~.~.agvInWay := vehicle.~.~.agvInWay - 1; --agv leaves the terminal
line
    end;
    vehicle.move(root.maintenance.repair_shop);
    vehicle.repairTime.rolldice;
    ref(freeRepairedVeh).methCall(vehicle.RepairTime, vehicle, typeB, status);
else
    -- vehicle waits at its current location for a while
    vehicle.stop;
    vehicle.repairStayTime.rolldice;
    ref(freeRepairedVeh).methCall(vehicle.RepairStayTime, vehicle, typeB, status);
end;
end; -- end of the method

```

Method Name: setDirection

Method's parent object: agvs_straight, and all other path objects

Description: This method is called when a vehicle enters a path object. It determines the new direction of the vehicle according to its current destination.

Code:

```
-----  
: string  
is  
local direction : string;  
    vehicle : object;  
do  
vehicle := @;  
if trackEntr.succ(1).void  
then          -- no successor  
    Message("noSucc","M","", "");  
    direction := "back";  
elseif trackEntr.pred(1).void  
then          -- no predecessor  
    Message("noPred","M","", "");  
    direction := "straight";  
else  
    if (current.uni_directional)  
    then  
        vehicle.driving_direction := "forward";  
        direction := "straight";  
    elseif (vehicle.driving_direction = "forward")  
    then  
        direction := "straight";  
    else  
        direction := "back";  
    end;  
end;  
if trackEntr.succ(1).void  
    OR trackEntr.succ(1).~.origin = .bat_charge_station  
    OR trackEntr.succ(1).~.origin = .maintenance  
then  
    direction := "back";  
    vehicle.driving_direction := "backward";  
end;  
result := direction;  
end; -- end of the method
```

Method Name: entrCtrl

Method's parent object: agvs_dock_station

Description: This method is called when a vehicle enters a dock station. It checks if there is a blockage in the next path section. If there is a blockage, the vehicle is awaited.

Code:

```
-----
is
local vehicle : object;
do
    vehicle := @;
if root.agvs_manager.zone_specific_speed
then
    -- set the specific zone speed
    old_speed := vehicle.fwspeed;
    vehicle.fwspeed := zone_speed;
end;
if (vehicle.freeVehicleStatus = "waitAtMachine") and
    (vehicle.waitingMachine = current) and
    (vehicle.empty) and (partReady.mu(1).void) and
    (vehicle.fd_dispoStatus = "free")
then -- vehicle waits at the station until there is order
    vehicle.move(vehicleSpur);
    if root.stats.veh_stats_on
    then
root.stats.setVehidleTimes((root.eventController.Simtime -
    vehicle.IdleMovingTime),
    vehicle, "moving");
    end;
    vehEntrTimeToSpur := root.eventController.Simtime;
    vehicle.fd_idleStatus := "atMachine";
    end;
if root.RouteMgr.zone_stats_on
then
    time_stats["vehicle", table_index] := vehicle;
    time_stats["enter_time", table_index] := root.eventController.simTime;
end;
if NOT (current = vehicle.Destination) -- station is not the destination
then -- call checkFront method
    checkFront;
end;
end; -- end of the method
```

Method Name: load_unload

Method's parent object: agvs_dock_station

Description: This method is called when a vehicle reaches to its local destination. It loads or unloads the parts to/from the vehicle.

Code:

```
-----
: time
is
local vehicle : object;
    duration : time;
    destL : list[object];
    seq_list : list;
    distance, old_Distance : real;
    m : integer;
    pointer : boolean;
    line : integer;
do
    vehicle := @;
    duration := 0;
    -- now there will be a load or unload action
    -- according to the status of vehicle and Partready
    if (vehicle.empty) and (not partReady.mu(1).void)
    then -- empty vehicle is loaded
        vehicle.loadTime.rolldice;
        duration := vehicle.loadTime;
    current.agv_load;
    elseif current = vehicle.Destination
    then -- vehicle is unloaded
        current.agv_unload;
        vehicle.loadTime.rolldice;
        duration := vehicle.loadTime;
    elseif (not partReady.mu(1).void) and (not vehicle.mu(1).void) and
        (vehicle.mu(1).globDestination = current.partReady.mu( 1
).globDestination)
    then -- vehicle is not empty but it is loaded if destinations are same
        current.agv_load;
        vehicle.loadTime.rolldice;
        duration := vehicle.loadTime;
    end;
if vehicle.numMu > 1
then
    -- if loads have different destinations then,
    -- according to the part destinations and the vehicle direction
    -- the vehicle choses the nearest station as its first dest.
```

```

destL.create;
from
m := vehicle.xDim;
until m = 0
loop
    destL.cursor := 1;
    if (not vehicle.mu(m).void)
        and (not destL.find(vehicle.mu(m).globDestination))
    then
        root.agvs_manager.rl_assignL.cursorX := 1;
        root.agvs_manager.rl_assignL.cursorY := 1;
        root.agvs_manager.rl_assignL.find(vehicle.mu(m).globDestination);
        destL.append(root.agvs_manager.rl_assignL["dockingStation",
root.agvs_manager.rl_assignL.cursorY]);
    end;
    m := m - 1;
end; -- end of loop
-- select the closest destination as first
old_distance := 1000.0; -- a big number
from
m := destL.dim;
until
m = 0
loop
    rl_destinationL.cursorX := 1;
    rl_destinationL.cursorY := 1;
    rl_destinationL.find(destL.read(m));
    distance := rl_destinationL["distance", rl_destinationL.cursorY];
    if distance > 0
        AND
        old_Distance > distance
    then
        vehicle.destination :=
            rl_destinationL["destinations", rl_destinationL.cursorY];
        old_Distance := distance;
    end;
    m := m - 1;
end;
end; -- if more than one load exist
-- RouteMgr settings#####
if root.RouteMgr.RouteMgr_active
then
    --root.RouteMgr.update_seq(vehicle); -- destination sequence is updated
    if root.RouteMgr.route_init_on
        AND
        (vehicle.~.~.origin = .agvs_dock_station OR

```

```

        vehicle.~.~.origin = .agvs_station)
    AND
    NOT (vehicle.destination.origin = .agvs_station)
    AND
    root.RouteMgr.dest_seq[vehicle.destination, vehicle.~.~].void
then -- initialize the dest_seq table
    seq_list.create;
    root.RouteMgr.dest_seq[vehicle.destination, vehicle.~.~] := seq_list;
    vehicle.origin_station := vehicle.~.~;
elseif NOT root.RouteMgr.route_init_on
    AND
    NOT root.RouteMgr.dest_seq[vehicle.destination, vehicle.~.~].void
then -- copy destination sequence for the vehicle
    root.RouteMgr.veh_dest_seq["sequenceList", vehicle].delete;
    root.RouteMgr.veh_dest_seq["sequenceList", vehicle] :=
        root.RouteMgr.dest_seq[vehicle.destination, vehicle.~.~];
end;
end;
-- end of Route settings#####
if root.stats.veh_stats_on
then
-- vehicle load statistics#####
if vehicle.numMu > 0
then
    --line := root.stats.loaded_veh_stats.Ydim + 1;
    --root.stats.loaded_veh_stats["vehicle", line] := vehicle;
    --root.stats.loaded_veh_stats["load", line] := vehicle.numMU;
    --if station_priority
    --then
    --    root.stats.loaded_veh_stats["priority", line] := "yes";
    --end;
if root.stats.UserDlg.dialogbox.getPopIndex("veh_stats_Pop2") = 4
    AND
    ((vehicle.vehicle_type = root.stats.veh_type)
    OR
    (root.stats.veh_type = "all"))
    AND
    (root.stats.priority = "all"
    OR (root.stats.priority = "y" AND vehicle.Mu(1).highPriority = true)
    OR (root.stats.priority = "n" AND vehicle.Mu(1).highPriority = false))
then -- update number of loads histogram
    root.stats.place_in_histogram(vehicle.numMU);
end;
end;
end; -- end of veh loaded stats#####
-- now the vehicle is sent away

```

```

if uni_directional = false
then
    pointer := set_direction;
else
    pointer := false; -- forward move
end;
vehicle.stop;
ref(vehCont).methCall(duration, pointer, vehicle);
--delete vehicle from the available vehicle list
root.agvs_manager.vd_freeVehl.cursorX := 1;
root.agvs_manager.vd_freeVehl.cursorY := 1;
if root.agvs_manager.vd_freeVehl.find(vehicle)
then
root.agvs_manager.vd_freeVehl.cutRow(root.agvs_manager.vd_freeVehl.cursor
Y);
    end;
end; -- end of the method

```

New methods in the new objects

New objects developed during this project are Battery Charger, Maintenance, Stats, RouteMgr, Build, and additional path objects. These new objects have many new methods in their structure. Some of these methods will be listed and explained below. A disk containing the new AGV library and two example models has also been provided with this report.

Method Name: buildMethod

Method's parent object: Build

Description: This method automatically creates a new AGV model. The layout data is read from a CAD drawing.

Code:

```

-----
is
local newObj, layer, direction, p_d_type, curve_type : string;
    CoordX, CoordY, coordX2, coordY2, dummy : real;
    lineNo, lines_count, path_points_count, i, j : integer;
    firstSide, secondSide : integer;

```

```

    nextLine : string;
    line_length, radius, first_degree, second_degree : real;
    check : boolean;
    pointObj, firstObj, secondObj : object;
do
delete_agvModel;
-- .AgvModel.openDialog;
check := false;
-- add managers to the model if none exists
if .AgvModel.numnodes = 0
then
    .eventcontroller.createBuildingBlock(.AgvModel, 10, 10);
    .agvs_manager.createBuildingBlock(.AgvModel, 60, 10);
    .ProdMgr.createBuildingBlock(.AgvModel, 100, 10);
    .stats.createBuildingBlock(.AgvModel, 10,50);
    .RouteMgr.createBuildingBlock(.AgvModel, 60, 50);
end;
-- delete current tables
lines.delete;
path_points.delete;
cpoints.delete;
directions.delete;
p_ds.delete;
intersections.delete;
curves.delete;
curves2.delete;
from
FileInterface.gotop;
lineNo := 1;
until FileInterface.eof
loop
    nextLine := FileInterface.readln;
    if nextLine = "ENTITIES"
    then
        check := true;
    end;
    if check
    then
        if nextLine = "INSERT"
        then
            -- read the type of layer
            FileInterface.gotoLine(lineNo + 2);
            lineNo := lineNo + 2;
            layer := FileInterface.readln;
            if layer = "CP"
            then

```

```

-- read X coordinate of the cp
FileInterface.gotoLine(lineNo + 4);
coordX := str_to_num(FileInterface.readln);
cpoints["x", cpoints.ydim + 1] := coordX;
--print coordX;
lineNo := lineNo + 4;
-- read Y coordinate of the cp
FileInterface.gotoLine(lineNo + 2);
coordY := str_to_num(FileInterface.readln);
cpoints["y", cpoints.ydim] := coordY;
--print coordY;
lineNo := lineNo + 2;
end;
if layer = "PD"
then
  -- read X coordinate of the p/d
  FileInterface.gotoLine(lineNo + 6);
  coordX := str_to_num(FileInterface.readln);
  p_ds["x", p_ds.ydim + 1] := coordX;
  -- register as a control point as well
  cpoints["x", cpoints.ydim + 1] := coordX;
  --print coordX;
  lineNo := lineNo + 6;
  -- read Y coordinate of the p/d
  FileInterface.gotoLine(lineNo + 2);
  coordY := str_to_num(FileInterface.readln);
  p_ds["y", p_ds.ydim] := coordY;
  -- register as a control point as well
  cpoints["y", cpoints.ydim] := coordY;
  --print coordY;
  lineNo := lineNo + 2;
  -- read type of the p/d
  FileInterface.gotoLine(lineNo + 16);
  p_d_type := FileInterface.readln;
  p_ds["p_d_type", p_ds.ydim] := p_d_type;
  --print coordY;
  lineNo := lineNo + 16;
end;
if layer = "CURVES"
then
  -- read X coordinate of the curve
  FileInterface.gotoLine(lineNo + 6);
  coordX := str_to_num(FileInterface.readln);
  curves2["x", curves2.ydim + 1] := coordX;
  --print coordX;
  lineNo := lineNo + 6;

```

```

-- read Y coordinate of the curve
FileInterface.gotoLine(lineNo + 2);
coordY := str_to_num(FileInterface.readln);
curves2["y", curves2.ydim] := coordY;
--print coordY;
lineNo := lineNo + 2;
-- read type of the p/d
FileInterface.gotoLine(lineNo + 16);
curve_type := FileInterface.readln;
curves2["curve_type", curves2.ydim] := curve_type;
--print coordY;
lineNo := lineNo + 16;
end;
if layer = "INTERSECTIONS"
then
  -- read X coordinate of the intersection point
  FileInterface.gotoLine(lineNo + 6);
  coordX := str_to_num(FileInterface.readln);
  intersections["x", intersections.ydim + 1] := coordX;
  --print coordX;
  lineNo := lineNo + 6;
  -- read Y coordinate of the intersection
  FileInterface.gotoLine(lineNo + 2);
  coordY := str_to_num(FileInterface.readln);
  intersections["y", intersections.ydim] := coordY;
  --print coordY;
  lineNo := lineNo + 2;
  -- read type of the intersection
  FileInterface.gotoLine(lineNo + 16);
  p_d_type := FileInterface.readln;
  intersections["intersection_type", intersections.ydim] := p_d_type;
  --print coordY;
  lineNo := lineNo + 16;
end;
if layer = "DIRECTIONS"
then
  -- read direction
  FileInterface.gotoLine(lineNo + 2);
  direction := FileInterface.readln;
  directions["direction", directions.ydim + 1] := direction;
  lineNo := lineNo + 2;
  -- read X coordinate of the direction
  FileInterface.gotoLine(lineNo + 2);
  coordX := str_to_num(FileInterface.readln);
  directions["x", directions.ydim] := coordX;
  --print coordX;

```

```

        lineNo := lineNo + 2;
        -- read Y coordinate of the direction
        FileInterface.gotoLine(lineNo + 2);
        coordY := str_to_num(FileInterface.readln);
        directions["y", directions.ydim] := coordY;
        --print coordY;
        lineNo := lineNo + 2;
    end;
end; -- end of INSERT
if nextLine = "LINE"
then
    lines_count := lines.Ydim + 1;
    -- read X1 coordinate of the line
    FileInterface.gotoLine(lineNo + 4);
    coordX := str_to_num(FileInterface.readln);
    lines["x1", lines_count] := coordX;
    lineNo := lineNo + 4;
    -- read y1 coordinate of the line
    FileInterface.gotoLine(lineNo + 2);
    coordY := str_to_num(FileInterface.readln);
    lines["y1", lines_count] := coordY;
    lineNo := lineNo + 2;
    -- read X2 coordinate of the line
    FileInterface.gotoLine(lineNo + 4);
    coordX2 := str_to_num(FileInterface.readln);
    lines["x2", lines_count] := coordX2;
    lineNo := lineNo + 4;
    -- read Y2 coordinate of the line
    FileInterface.gotoLine(lineNo + 2);
    coordY2 := str_to_num(FileInterface.readln);
    lines["y2", lines_count] := coordY2;
    lineNo := lineNo + 2;
    -- determine vertical or horizontal line type
    -- determine the length and offset point
    if coordY = coordY2
    then
        if coordX2 < coordX
        then
            dummy := coordX;
            coordX := coordX2;
            coordX2 := dummy;
        end;
    else
        if coordY2 < coordY
        then
            dummy := coordY;

```

```

        coordy := coordy2;
        coordy2 := dummy;
    end;
end;
lines["x1", lines_count] := coordX;
lines["y1", lines_count] := coordY;
lines["x2", lines_count] := coordX2;
lines["y2", lines_count] := coordy2;
end; -- end of LINE insertion
if nextLine = "ARC"
then
    lines_count := curves.Ydim + 1;
    -- read X coordinate of the arc center
    FileInterface.gotoLine(lineNo + 4);
    coordX := str_to_num(FileInterface.readLine);
    curves["centerX", lines_count] := coordX;
    lineNo := lineNo + 4;
    -- read y coordinate of the arc center
    FileInterface.gotoLine(lineNo + 2);
    coordY := str_to_num(FileInterface.readLine);
    curves["centerY", lines_count] := coordY;
    lineNo := lineNo + 2;
    -- read the radius
    FileInterface.gotoLine(lineNo + 4);
    radius := str_to_num(FileInterface.readLine);
    curves["radius", lines_count] := radius;
    lineNo := lineNo + 4;
    -- read and calculate the degree of the arc
    FileInterface.gotoLine(lineNo + 2);
    first_degree := str_to_num(FileInterface.readLine);
    lineNo := lineNo + 2;
    FileInterface.gotoLine(lineNo + 2);
    second_degree := str_to_num(FileInterface.readLine);
    curves["degree", lines_count] := abs(second_degree - first_degree);
    lineNo := lineNo + 2;
    -- calculate the start and end points of the curve
    if abs(second_degree - first_degree) = 90
    OR
    abs(second_degree - first_degree) = 270
    then
        if first_degree = 180 AND second_degree = 270
        OR
        first_degree = 270 AND second_degree = 180
        then
            curves["firstX", lines_count] := coordX - radius;
            curves["firstY", lines_count] := coordY;

```

```

        curves["secondX", lines_count] := coordX;
        curves["secondY", lines_count] := coordY - radius;
    end;
if first_degree = 270 AND second_degree = 0
    OR
    first_degree = 0 AND second_degree = 270
then
    curves["firstX", lines_count] := coordX;
    curves["firstY", lines_count] := coordY - radius;
    curves["secondX", lines_count] := coordX + radius;
    curves["secondY", lines_count] := coordY;
end;
if first_degree = 90 AND second_degree = 0
    OR
    first_degree = 0 AND second_degree = 90
then
    curves["firstX", lines_count] := coordX + radius;
    curves["firstY", lines_count] := coordY;
    curves["secondX", lines_count] := coordX;
    curves["secondY", lines_count] := coordY + radius;
end;
if first_degree = 90 AND second_degree = 180
    OR
    first_degree = 180 AND second_degree = 90
then
    curves["firstX", lines_count] := coordX;
    curves["firstY", lines_count] := coordY + radius;
    curves["secondX", lines_count] := coordX - radius;
    curves["secondY", lines_count] := coordY;
end;
end;
end; -- end if arc
end; -- end if check
lineNo := lineNo + 1;
end;
-- first, deal with intersection points (traffics) and the lines of interest
place_intersections;
-- then, place the curves and the lines that are connected to them
place_curves;
-- later, place the destinations
place_pds;
-- connect the unconnected lines
connect_lines;
.AgvModel.openDialog;
end; -- end of the method

```

Method Name: setVehTimes

Method's parent object: Stats

Description: This method organizes the time statistics for vehicles.
Every change of a vehicle status updates the tables in the Stats object.

Code:

```
-----  
(time_elapsed : time ; vehicle : object ; status : string)  
is  
local oldTime : time;  
do  
veh_stats.cursorX := 1;  
veh_stats.cursorY := 1;  
if veh_stats.find(vehicle)  
then  
  if status = "free"  
  then  
    oldTime := veh_stats["idle_time", veh_stats.cursorY];  
    veh_stats["idle_time", veh_stats.cursorY] := time_elapsed + oldTime;  
  end;  
  if status = "assigned"  
  then  
    oldTime := veh_stats["assigned", veh_stats.cursorY];  
    veh_stats["assigned", veh_stats.cursorY] := time_elapsed + oldTime;  
  end;  
  if status = "occupied"  
  then  
    oldTime := veh_stats["occupied", veh_stats.cursorY];  
    veh_stats["occupied", veh_stats.cursorY] := time_elapsed + oldTime;  
  end;  
  if status = "failed"  
  then  
    oldTime := veh_stats["failed", veh_stats.cursorY];  
    veh_stats["failed", veh_stats.cursorY] := time_elapsed + oldTime;  
  end;  
  if status = "charging"  
  then  
    oldTime := veh_stats["charging", veh_stats.cursorY];  
    veh_stats["charging", veh_stats.cursorY] := time_elapsed + oldTime;  
  end;  
  if status = "blocked"  
  then  
    oldTime := veh_stats["blocked", veh_stats.cursorY];  
    veh_stats["blocked", veh_stats.cursorY] := time_elapsed + oldTime;  
  end;  
end;  
end; -- end of the method
```

Method Name: setMachineTimes

Method's parent object: Stats

Description: This method organizes the time statistics for machines. Every change of a machine status updates the tables in the Stats object.

Code:

```
-----  
(time_elapsed : time ; machine : string ; status : string)  
is  
local oldTime : time;  
do  
    oldTime := machine_stats[status, machine];  
    machine_stats[status, machine] := time_elapsed + oldTime;  
end; -- end of the method
```

Method Name: stopVehicle

Method's parent object: RouteMgr

Description: This method is called when a vehicle is blocked. It awaits the vehicle until the path is clear.

Code:

```
-----  
(vehicle : object)  
is  
local old_status : string;  
do  
if NOT blockage  
then  
    blockage := true;  
    blocked_vehicles := blocked_vehicles + 1;  
else  
    blocked_vehicles := blocked_vehicles + 1;  
end;  
if blocked_vehicles > max_q_length  
then  
    max_q_length := blocked_vehicles;  
end;  
    -- vehicle is stopped and the front is checked  
    -- by calling the timeLoop method  
    vehicle.stop;  
    old_status := vehicle.fd_dispoStatus;  
    ref(timeLoop).methCall(1, vehicle, old_status);
```

```

    root.agvs_manager.vd_changeStatus(vehicle, "blocked");
    root.agvs_manager.vd_tableUpdate(vehicle, "newTime");
end; -- end of the method

```

Method Name: timeLoop

Method's parent object: RouteMgr

Description: While the vehicle is waiting in front of a blocked section, this method updates the statistics regarding the blockage and the vehicle status.

Code:

```

-----
(vehicle : object ; old_status : string)
is
local block_time : real;
do
if vehicle.fd_dispoStatus /= "charging"
    AND
    (vehicle.fd_dispoStatus /= "failed")
then
    if ((vehicle.driving_direction = "forward")
        AND
        (NOT vehicle.~.succ(1).void)
        AND
        (NOT vehicle.~.succ(1).empty))
    or
        ((vehicle.driving_direction = "backward")
        AND
        (NOT vehicle.~.pred(1).void)
        AND
        (NOT vehicle.~.pred(1).empty))
    then
        ref(timeLoop).methCall(1, vehicle, old_status);
    else
        -- release the vehicle
        vehicle.continue;
        root.agvs_manager.vd_changeStatus(vehicle, old_status);
        root.agvs_manager.vd_tableUpdate(vehicle, "newTime");
        blocked_vehicles := blocked_vehicles - 1;
        if blocked_vehicles = 0
        then

```

```

        blockage := false;
    end;
-- update zone stats#####
    if zone_stats_on
    then
        if NOT (vehicle.~.~.origin = .agvs_br2)
            AND
            NOT (vehicle.~.~.origin = .agvs_junction2)
            AND
            NOT (vehicle.~.~.origin = .agvs_tl_br)
            AND
            NOT (vehicle.~.~.origin = .maintenance)
        then
            block_time := root.eventController.simTime -
                vehicle.~.~.time_stats["enter_time", vehicle.~.~.table_index];
            vehicle.~.~.time_stats["block_time", vehicle.~.~.table_index] := block_time;
        end;
        if (vehicle.~.~.origin = .agvs_tl_br)
            AND
            vehicle.~.~.name = "track_s"
        then
            vehicle.~.~.time_stats_s["vehicle", vehicle.~.~.table_index_s] := vehicle;
            block_time := root.eventController.simTime -
                vehicle.~.~.time_stats_s["enter_time",
vehicle.~.~.table_index_s];
            vehicle.~.~.time_stats_s["block_time", vehicle.~.~.table_index_s] :=
block_time;
            vehicle.~.~.table_index_s := vehicle.~.~.table_index_s + 1;
        end;
        if (vehicle.~.~.origin = .agvs_br2)      OR
            (vehicle.~.~.origin = .agvs_junction2)
        then
            if vehicle.~.~.name = "track_R"
            then
                vehicle.~.~.time_stats_R["vehicle", vehicle.~.~.table_index_R] :=
                    vehicle;
                block_time := root.eventController.simTime -
                    vehicle.~.~.time_stats_R["enter_time",
vehicle.~.~.table_index_R];
                vehicle.~.~.time_stats_R["block_time", vehicle.~.~.table_index_R] :=
block_time;
                vehicle.~.~.table_index_R := vehicle.~.~.table_index_R + 1;
                if zone_direction = "straight"
                then
                    place_in_histogram(block_time);
                end;
            end;
        end;
    end;

```

```

elseif vehicle.~.name = "track_L"
then
    vehicle.~.time_stats_L["vehicle", vehicle.~.table_index_L]:=
        vehicle;
    block_time := root.eventController.simTime -
        vehicle.~.time_stats_L["enter_time",
vehicle.~.table_index_L];

    vehicle.~.time_stats_L["block_time", vehicle.~.table_index_L] :=
block_time;
    vehicle.~.table_index_L := vehicle.~.table_index_L + 1;
    if zone_direction = "left"
    then
        place_in_histogram(block_time);
    end;
end;
if (vehicle.~.origin = .agvs_station)
then
    vehicle.~.time_stats["vehicle", vehicle.~.table_index]:=
        vehicle;
    vehicle.~.table_index := vehicle.~.table_index + 1;
end;
if (vehicle.~ = selected_zone)
    AND
    NOT ((vehicle.~.origin = .agvs_br2) OR (vehicle.~.origin =
.agvs_junction2))
    then
        place_in_histogram(block_time);
    end;
if (vehicle.~.max_block["max_block_time", 1] <
    block_time)
    then
        vehicle.~.max_block["max_block_time", 1] := block_time;
        vehicle.~.max_block["vehicle", 1] := vehicle;
    end;
if (max_block["max_block_time", 1] < block_time)
    then
        max_block["max_block_time", 1] := block_time;
        max_block["vehicle", 1] := vehicle;
        max_block["zone", 1] := vehicle.~.;
    end;
end;
end;
end;
end; -- end of the method

```